

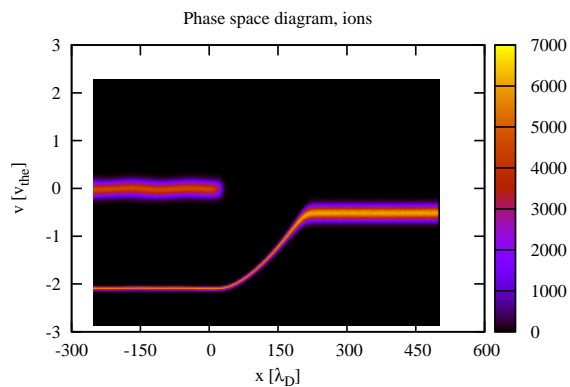
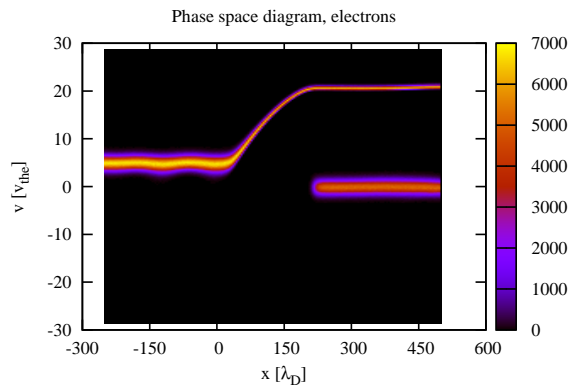
**UNIVERSITY OF OSLO**  
**Department of Physics**

# **Numerical Simulations of Double Layers in Plasmas**

Master thesis

Vegard Lundby  
Rekaa

March 2009



*The figures on the front page are phase space diagrams taken from one of the simulations performed as a part of this Master's thesis. A fictive mass ratio of  $m_i/m_e = 100$  is used.*





# Abstract

In this thesis, a numerical study of double layers in plasma is presented. Double layers are structures consisting of two oppositely charged space charge layers, creating a finite change in electrostatic potential over the double layer. The motivation for the topic is found in the general interest for magnetospheric physics, where double layers are known to accelerate particles into the ionosphere, creating the aurora, as seen at higher latitudes.

In the first part of this thesis, theories and observations of double layers as a plasma physics phenomena are studied. Two double layer related phenomena, the Buneman instability and adiabatic cooling, are presented and their expected influence on double layers given. Then, through the introduction of BGK-solutions and the Water-bag model, a one dimensional, time stationary, electrostatic model of double layers is formed.

In the second part of this thesis, the theories discussed in the first part are implemented into a one dimensional numerical model of double layers. The numerical model emphasizes strong (i.e. Buneman regime) double layers, and their existence is investigated for a large number of different plasma and boundary conditions.

The results of the numerical model can confirm important observations made by previous numerical and experimental studies, like the scaling law, Bohm criterion and presence of two-stream (i.e. Buneman) instabilities. The present numerical model improves on these studies by introducing simulations with varying combinations of boundary conditions, and a wider selection of plasma conditions for which double layers have been simulated.

For the numerical studies of this thesis, a one dimensional Particle-in-Cell code was designed and developed. A thorough description of the program is given in the main part of this thesis, and the source code is shown in the appendix.



# Acknowledgements

First of all, I would like to thank my supervisors, Profs. Hans Pécseli and Jan Trulsen for all your guidance, patience and good spirit while working with me these two years. You have inspired me to push my own boundaries and complete tasks I did not believe was possible before we started. It has been a great pleasure working with you!

Secondly I would like to thank my dear Ellen. You have given me encouragement and guidance through all the discussions we've had, and you have shown patience with my early mornings and late evenings at the office, and for taking care of our household all by yourself so that I could spend more time working on this thesis.

I would also like to thank those who helped me with proof reading in the final stages of these two years; Anders Rekaa, Ann Blomberg, Ellen Osmundsen, Wojciech Miloch and Brian Schratz. Last but not least, a thank you to all my co-students and employees at the group for Plasma- and Space physics and the Institute of Astrophysics for your encouragement along the way, and for an inspiring working environment.

And, as a final comment: Thanks to the three large bill boards in my office. How could I have gotten such good overview of my results as I did without you?!





# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 The Basics of Plasma Physics</b>	<b>3</b>
2.1 Plasma in nature and experiments . . . . .	3
2.2 Theoretical models . . . . .	4
2.2.1 Single particle description . . . . .	6
2.2.2 Kinetic description . . . . .	6
2.2.3 Fluid description . . . . .	7
2.3 Basic quantities . . . . .	8
2.3.1 Thermal velocity . . . . .	8
2.3.2 Debye length . . . . .	8
2.3.3 Plasma frequency . . . . .	9
2.3.4 The plasma parameter . . . . .	10
2.4 Plasma physics through numerical simulations . . . . .	10
2.4.1 The Particle-in-Cell method . . . . .	11
2.4.2 The Vlasov method . . . . .	12
2.4.3 Fluid description . . . . .	12
2.4.4 Boundary conditions . . . . .	13
<b>3 The Physics of Double Layers</b>	<b>15</b>
3.1 Double layers in nature and experiments . . . . .	16
3.2 Particle interactions . . . . .	16
3.3 Properties and related phenomena . . . . .	17
3.3.1 The scaling law . . . . .	17
3.3.2 The Bohm criterion . . . . .	18
3.3.3 The Langmuir condition . . . . .	18
3.3.4 Buneman instability . . . . .	19
3.3.5 Adiabatic cooling . . . . .	22
3.4 A summary of characteristic . . . . .	25
3.4.1 Strong double layers . . . . .	25
3.4.2 Weak double layers . . . . .	26
3.5 Theoretical models in one dimension . . . . .	27
3.5.1 BGK solutions . . . . .	27
3.5.2 Probability density functions of a double layer . . . . .	28
3.5.3 Water-bag models . . . . .	29
3.5.4 The Sagdeev potential . . . . .	30
3.5.5 Examples in one dimension . . . . .	31

3.6	Numerical studies of double layers . . . . .	35
<b>4</b>	<b>Numerical model</b>	<b>39</b>
4.1	General assumptions and approximations . . . . .	39
4.2	Dimensionless equations and quantities . . . . .	39
4.2.1	Equations of motion . . . . .	40
4.2.2	Field equations . . . . .	40
4.2.3	Flux density . . . . .	41
4.3	Discretized equations . . . . .	41
4.3.1	Equations of motion . . . . .	41
4.3.2	Field equations . . . . .	42
4.4	Statistical methods . . . . .	43
4.4.1	The Inversion method . . . . .	43
4.4.2	The Box-Muller algorithm . . . . .	44
4.5	Simulation routines . . . . .	44
4.5.1	The Particle-in-Cell method . . . . .	45
4.5.2	Poisson's equation . . . . .	46
4.5.3	Electric field . . . . .	52
4.5.4	Equations of motion/Leapfrog algorithm . . . . .	52
4.5.5	Flux density . . . . .	52
4.6	Equilibrium plasma . . . . .	53
4.7	Double layer . . . . .	54
4.7.1	Initial probability density functions . . . . .	54
4.8	Diagnostic routines . . . . .	58
4.8.1	Charge density, potential and electric field . . . . .	58
4.8.2	Density profiles . . . . .	59
4.8.3	Flux . . . . .	59
4.8.4	Phase space diagrams . . . . .	60
4.8.5	Fourier analysis . . . . .	60
<b>5</b>	<b>Results</b>	<b>65</b>
5.1	Reference simulation . . . . .	65
5.2	Stability . . . . .	70
5.2.1	Boundary conditions dependencies . . . . .	70
5.2.2	Parameter dependencies . . . . .	73
5.2.3	Scaling law . . . . .	74
5.3	Harmonic oscillations . . . . .	75
5.3.1	Temporal harmonic oscillations . . . . .	75
5.3.2	Spatial harmonic oscillations . . . . .	77
5.4	Phase space vortices . . . . .	79
<b>6</b>	<b>Discussions and Conclusions</b>	<b>83</b>
6.1	Boundary conditions . . . . .	83
6.2	Plasma parameters . . . . .	84
6.3	Scaling law . . . . .	86
6.4	Buneman instability . . . . .	86
6.5	Summary . . . . .	87
6.6	Future work . . . . .	88
	<b>Bibliography</b>	<b>88</b>

<b>A</b>	<b>Source code</b>	<b>93</b>
A.1	Simulation code	94
A.1.1	Compilation and execution	94
A.1.2	kode.h	94
A.1.3	main.cpp	95
A.1.4	1D_plasma_simulations.cpp	96
A.1.5	initializeDL.cpp	97
A.1.6	phi_integrals.cpp	100
A.1.7	flux.cpp	102
A.1.8	fourier.cpp	103
A.1.9	diagnostics.cpp	105
A.2	Gnuplot scripts	109
A.2.1	Output to .ps script	109
A.2.2	Output to multiple .eps files script	109
A.2.3	script.header.p	109
A.2.4	script.footer.p	109
A.2.5	results.fields.p	109
A.2.6	results.carpet.p	110
A.2.7	results.density.p	110
A.2.8	results.flux.p	110
A.2.9	results.vdf.p	111
A.2.10	results.phasespace.p	111
A.2.11	results.power.p	111



# List of Figures

1.1	<i>Schematic illustration of a double layer.</i>	1
2.1	<i>Sketch of the Wendelstein 7-AS stellarator fusion device</i>	4
2.2	<i>Sketch of ITER fusion device</i>	5
2.3	<i>Illustration shows an example of how a particle in position <math>x</math> is weighted down to spatial grid points with positions <math>x_i</math> and <math>x_{i+1}</math>. The height of the triangular shaped particle represents the weighting of the particle.</i>	12
3.1	<i>Diagram for analyzing stability conditions of the Buneman instability</i>	21
3.2	<i>Double layer creation by the Buneman instability [Belova et al., 1980]</i>	21
3.3	<i>Variation of the ion distribution function with normalized plasma potential</i>	24
3.4	<i>Variation of the normalized drift velocity with normalized plasma potential leap.</i>	24
3.5	<i>Variation of the normalized effective temperature of the distribution function with normalized plasma potential leap.</i>	24
3.6	<i>Typical characteristics of strong and weak double layers [Schamel, 1986]</i>	25
3.7	<i>Assuming a <math>\tanh(x)</math> of the double layer potential</i>	32
3.8	<i>Phase space representation of the double layer probability density functions</i>	33
3.9	<i>Double layer particle density profiles</i>	33
3.10	<i>Water-bag model contours of a double layer</i>	34
3.11	<i>Sagdeev potential of a double layer</i>	35
4.1	<i>Illustration of the Leapfrog algorithm</i>	42
4.2	<i>Example of the transform performed by the Box-Muller algorithm</i>	44
4.3	<i>Time cycle of present simulations</i>	45
4.4	<i>The Particle-in-Cell method's boundary conditions</i>	46
4.5	<i>Illustration of the Red-Black algorithm</i>	48
4.6	<i>Illustration of the Multigrid method</i>	49
4.7	<i>An example of the Poisson solvers Gauss-Seidel, Red-Black and Multigrid.</i>	50
4.8	<i>A comparison of the Poisson solvers Gauss-Seidel, Red-Black and Multigrid</i>	51
4.9	<i>Double layer probability density functions of potential <math>\phi</math> and velocity <math>v</math></i>	55
4.10	<i>Initialization of a potential profile <math>\phi(x)</math> from the charge density <math>\rho(\phi)</math></i>	57
4.11	<i>The electrostatic potential with the corresponding PDFs for a double layer, as initialized in present simulations.</i>	57
4.12	<i>Example plots of the diagnostic routines for monitoring temporal variations in fields</i>	58
4.13	<i>Example plots of the diagnostic routine monitoring particle density profiles</i>	59
4.14	<i>Example plot of the diagnostic routine monitoring flux</i>	60
4.15	<i>Example plots of the diagnostic routine creating phase space diagrams</i>	61
4.16	<i>The window function used in Fourier analysis</i>	62

4.17	<i>Example field subject to windowing and Fourier analysis . . . . .</i>	62
4.18	<i>Power spectrum of the example field subject to Fourier analysis . . . . .</i>	63
4.19	<i>Example plot of the diagnostic routine monitoring temporal evolution of oscillations . . . . .</i>	63
5.1	<i>Stacked field plots of the reference simulation . . . . .</i>	67
5.2	<i>Temporal evolution of oscillations and density profiles of the reference simulation . . . . .</i>	68
5.3	<i>Flux and phase space diagrams of the reference simulation . . . . .</i>	69
5.4	<i>Field stack plots of the simulation <math>L = 750\lambda_{De}</math>, <math>\phi_{DL} = 50</math>, <math>m_i/m_e = 100</math> and <math>U = 5</math> . . . . .</i>	71
5.5	<i>Field stack plots of the simulation <math>L = 750\lambda_{De}</math>, <math>\phi_{DL} = 500</math>, <math>m_i/m_e = 100</math> and <math>U = 5</math>. . . . .</i>	72
5.6	<i>Lifetime of the simulations as a function of double layer width, <math>L</math>, and potential leap, <math>\phi_{DL}</math> . . . . .</i>	73
5.7	<i>Lifetime of the simulations as a function of drift velocity, <math>U</math>, and potential leap, <math>\phi_{DL}</math> . . . . .</i>	74
5.8	<i>The scaling law compared to present results . . . . .</i>	75
5.9	<i>Temporal evolution of potential leap, <math>\phi_{DL}(t)</math>, as a function of time <math>t</math> and initial potential leap, <math>\phi_{DL}(t = 0)</math> . . . . .</i>	76
5.10	<i>Angular frequency of temporal oscillations as function of potential leap <math>\phi_{DL}</math> and drift velocity, <math>U</math> . . . . .</i>	77
5.11	<i>Temporal evolution of oscillations for the simulation <math>L = 500</math>, <math>\phi_{DL} = 500</math>, <math>m_i/m_e = 100</math> and <math>U = 3</math>. . . . .</i>	78
5.12	<i>Linear fit to the variation of wave numbers of spatial oscillations . . . . .</i>	79
5.13	<i>Temporal evolution of the oscillations for the simulation <math>L = 125</math>, <math>\phi_{DL} = 400</math>, <math>m_i/m_e = 100</math> and <math>U = 5</math> . . . . .</i>	80
5.14	<i>Field stack plots and phase space diagrams for <math>L = 125</math>, <math>\phi_{DL} = 400</math>, <math>m_i/m_e = 100</math>, <math>U = 5</math> and D-D boundary conditions. . . . .</i>	81

# List of Tables

3.1	<i>List of previous numerical studies of double layers . . . . .</i>	36
3.2	<i>Boundary conditions of the potential in present simulations . . . . .</i>	37
4.1	<i>Number of operations in the Multigrid method on a one dimensional grid of size <math>N</math> . . . . .</i>	50
5.1	<i>List of the variable parameters of the present simulations . . . . .</i>	65
5.2	<i>Two common oscillation modes from the present results, listed for different double layer widths <math>L</math> . . . . .</i>	78
5.3	<i>The combinations of the parameters double layer width, <math>L</math>, potential, <math>\phi_{DL}</math>, mass ratio, <math>m_i/m_e</math> and drift velocity, <math>U</math>, that has been simulated in the present studies . . . . .</i>	82





# Chapter 1

## Introduction

Plasma is a partially or fully ionized gas, in which a proportion of the electrons are free to move without being bound to an atom or molecule. The independent movement of the charged particles makes the plasma electrically conductive and therefore subject to electromagnetic forces. As these properties are not present in either solids, liquids or gases, plasma is referred to as the fourth state of matter [Chen, 1984].

Double layer is a name given to certain structures in plasmas consisting of two oppositely charged parallel sheaths in a plasma. The sheaths form a localized electric field between them, with a corresponding finite change in the electrostatic potential  $\phi_{DL}$  [Raadu, 1989]. Double layers are often found in coexistence with currents flowing parallel to the electric field. In magnetospheric physics, these structures are mostly recognized by 1) particles accelerated through the double layers and 2) *in situ* measurements of the electrostatic potential leap. An example of a typical double layer structure is illustrated in figure 1.1, with corresponding charge density, electric field and potential.

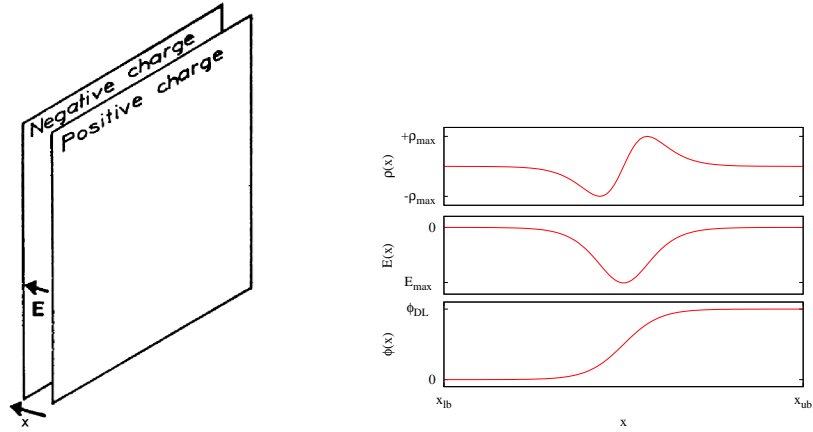


Figure 1.1: *Left panel: a schematic illustration of a DL ( Courtesy of: [Block, 1978]). Right panel: Typical examples of charge density, electrostatic potential and electric field of a DL.*

Double layers are thought to be created either in sheaths where two plasmas of different properties merge, or plasmas dominated by currents. Double layers created under such conditions are respectively called *weak* and *strong*. In this context, *weak* and *strong* refer to the energy gained or lost by a particle passing through the double layer,  $q\phi_{DL}$ ,

compared to its thermal energy  $\kappa T$ . Strong and weak double layers are seen to have highly different characteristics, which will be discussed in the main part of the thesis.

Double layers are also found to be a realization of the theoretical class of solutions known as "BGK-solutions" [Bernstein et al., 1957]. The BGK-solutions describe non-linear stationary solutions of the Vlasov-Poisson set of equations. Only three of these solutions have been observed in nature and experiments: Double layers, ion- and electron holes [Schamel, 1986]. BGK-solutions form the basis of the Water-bag theory [Davidson, 1972], where the plasma is prescribed by stepwise constant probability density functions. The Water-bag model allows for a simple, yet powerful, visualization of double layers (and similar BGK-solutions) and the behaviour of particle species involved.

The Buneman instability is found to have an important role in double layer dynamics [Belova et al., 1980; Omura et al., 2008], both as a creation mechanism (in the case of *strong* double layers) and the destruction of an already existing double layer. The Buneman instability has also been observed to saturate into ion- and electron holes in plasmas containing double layers.

This thesis introduces a theoretical and numerical study of double layers in plasmas. Characteristics of the particle species which are present in the double layer are defined and existing theories and observations describing the double layers are given. Based on these, the numerical model has been developed and presented in this thesis, followed by the results created from the simulations. These results are then compared to the theories discussed earlier.

The numerical model is a one-dimensional, Particle-in-Cell code, simulating double layers in plasmas consisting of two particle species; electrons and singly charged positive ions. The existence of double layers is tested for different plasma conditions, defined through variations in typical plasma parameters and the boundary conditions of the simulations. The model is designed for strong double layers, assuming no effects of magnetic fields.

## Chapter 2

# The Basics of Plasma Physics

In the present chapter, basic properties, theories and quantities in plasma physics are presented, then the theories are rephrased into general numerical models. This chapter thus gives the basic knowledge needed to perform and understand numerical simulations of plasma physics in general and those needed to perform the double layer simulations of the this thesis. The theory and properties of double layers will be discussed in chapter 3.

### 2.1 Plasma in nature and experiments

The earliest recording of plasma studies dates back to 1879 and Sir William Crookes. His experiment, the Crookes Tube, was simply a cathode ray in a vacuum tube of glass. Crookes observed that gas inside the tube started to glow when struck by the cathode rays, thus giving the name *radiant matter* to the glowing gas inside. It was first in the late 1920's that the name *plasma* was introduced by Irving Langmuir [Langmuir, 1929]. He was working with gas filaments<sup>1</sup> in vacuum and the emission of charged particles from hot filaments (thermionic emission) which led him to discover waves in the electron density, known today as *Langmuir waves*. The name *plasma* refers to the resemblance Langmuir thought it had to blood plasma.

Today, studies of plasma media are motivated by the wide spread occurrence of plasmas in the Earth's near and distant space environments, where most of the classical matter is in the plasma state ( $> 99\%$ ). In our Solar system, for instance, most of the matter is in the Sun, where it can safely be taken to be fully ionized. The motivation for the present work is found in the general interest for double layers in the magnetosphere (not necessarily that of the Earth), where acceleration of charged particles will occur. Double layers are also found in industrial devices, e.g. high voltage breakers, and seemingly the first studies of double layers were motivated by observations in such devices.

It should however be mentioned that one of the greatest incentives for modern plasma studies is concentrated around fusion plasma physics [Wilhelmsson, 2000]. It has been suggested that reactors based on nuclear fusion of light particles (Hydrogen, Deuterium, Tritium, etc.) can ultimately be the solution of mankind's energy demands. It is generally agreed that modern plasma-fusion studies began with an international Geneve-conference in 1956. Many different experimental devices have been considered [Bishop, 1958], where some early suggestions were based on pinch-type plasma discharges. More modern, and also more advanced, concepts are based on the Stellerator device, with a new Wendel-

---

<sup>1</sup>A slender or thread-like structure



Figure 2.1: A sketch of the magnetic coils in the future stellarator fusion device Wendelstein 7-AS (Max-Planck Institute in Greifswald, Germany) Source of figure: [www.efda.org, 2008].

stein 7AS-stellarator (see figure 2.1) under construction at the Max-Planck Institute in Greifswald, Germany. Most modern experiments are, however, based on the Tokamak concept, where the latest (and largest) device ITER (see figure 2.2) is under construction in Cadarache, in southern France. The Tokamak device can be seen as a transformer, where the secondary winding is constituted by the plasma which in turn is confined by a toroidal magnetic field. Neither the Stellarator, nor the Tokamak will be discussed further in this thesis.

## 2.2 Theoretical models

The charged particles of a plasma will, in the presence of a magnetic field  $\mathbf{B}$  and electric field  $\mathbf{E}$ , experience the Lorentz force,

$$\mathbf{F} = q_j(\mathbf{E}(\mathbf{X}_j, t) + \mathbf{V}_j(t) \times \mathbf{B}(\mathbf{X}_j, t)) , \quad (2.1)$$

where  $\mathbf{X}_j = \mathbf{X}_j(t)$ ,  $\mathbf{V}_j = \mathbf{V}_j(t)$  and  $q_j$  are the position, velocity and charge of particle  $j$ , respectively. The electric and magnetic fields, which are external and/or self-consistent, are found with Maxwell's equations

$$\nabla \cdot \mathbf{E} = \frac{\rho_e}{\epsilon_0} , \quad (2.2)$$

$$\nabla \cdot \mathbf{B} = 0 , \quad (2.3)$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} , \quad (2.4)$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} , \quad (2.5)$$

where  $\rho_e$  and  $\mathbf{j}$  are the plasma charge and current densities, while  $\epsilon_0$  and  $\mu_0$  are electric susceptibility and magnetic permeability, respectively. The electric field can be expressed in terms of scalar and vector potentials  $\mathbf{E} = -\nabla\phi - \partial\mathbf{A}/\partial t$ . In many cases relevant for plasma physics, the waves are assumed purely electrostatic and the vector potential  $\mathbf{A}$

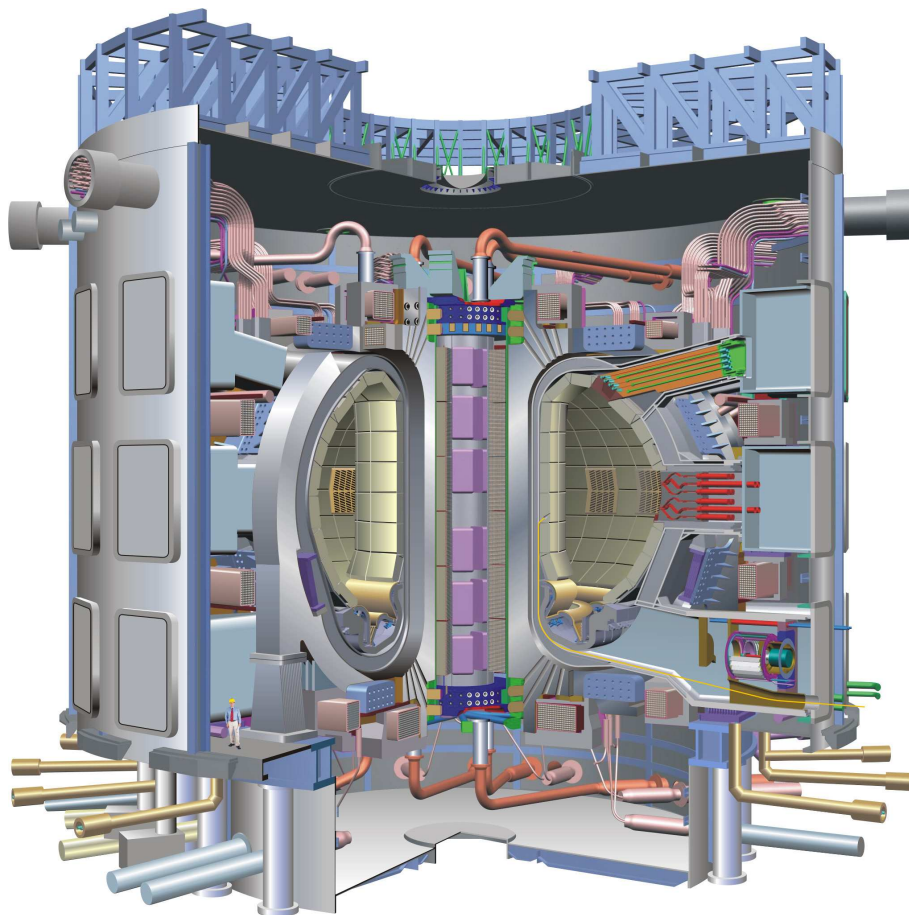


Figure 2.2: *An illustration of the future fusion device ITER (Cadarache, in southern France) based on earlier Tokamak devices (e.g. JET in Culham, Oxford, England). Note the small man inserted as a scale in the lower left part of the figure! Source of figure: [www.iter.org, 2008].*

can be omitted, thereby simplifying Poisson's equation (2.2) to

$$\nabla^2 \phi = -\frac{\rho_e}{\epsilon_0} . \quad (2.6)$$

The forces acting on a plasma (the Lorentz' force, pressure forces, etc.) can be evaluated on different levels of complexity. Three of these levels are single particle, kinetic or fluid descriptions of the plasma. The theory and properties of these three descriptions are given in the following sections.

### 2.2.1 Single particle description

To describe the plasma through the discrete nature of the particles, the equations to be solved together with Maxwell's equations are

$$\frac{d\mathbf{X}_j}{dt} = \mathbf{V}_j , \quad (2.7)$$

$$\frac{d\mathbf{V}_j}{dt} = \frac{q_j}{m_j}(\mathbf{E}(\mathbf{X}_j, t) + \mathbf{V}_j \times \mathbf{B}(\mathbf{X}_j, t)) , \quad (2.8)$$

$$\rho_e = \rho_e(\mathbf{x}, t) = \int \sum_s q_s N(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} , \quad (2.9)$$

$$\mathbf{j} = \mathbf{j}(\mathbf{x}, t) = \int \sum_s q_s \mathbf{V}_j P(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} , \quad (2.10)$$

where  $\mathbf{x}$  and  $\mathbf{v}$  are coordinates of the spatial-velocity phase space and

$$N(\mathbf{x}, \mathbf{v}, t) = \sum_j \delta(\mathbf{v} - \mathbf{V}_j) \delta(\mathbf{x} - \mathbf{X}_j) ,$$

$$P(\mathbf{x}, \mathbf{v}, t) = \sum_j \mathbf{V}_j \delta(\mathbf{v} - \mathbf{V}_j) \delta(\mathbf{x} - \mathbf{X}_j) ,$$

have been introduced as particle density ( $N(\mathbf{x}, \mathbf{v}, t)$ ) and particle flux ( $P(\mathbf{x}, \mathbf{v}, t)$ ) in phase space, where  $\delta(\mathbf{r})$  is the Dirac delta function<sup>2</sup>.

To describe a plasma consisting of  $\mathcal{N}$  particles by brute force, the above given relations have to be solved for each particle  $j$ , calculating the particle's interaction with the remaining  $\mathcal{N} - 1$  particles. By summing over  $j$ , it is easy to find that equations (2.7)-(2.10) must be solved  $\mathcal{N}^2$  times. The complexity of the problem can be expressed with the notation  $\mathcal{O}(\mathcal{N}^2)$ ; that the number of operations necessary to solve the problem are proportional to  $\mathcal{N}^2$ .

It is impossible to give a direct solution to a problem of such large proportions numerically, let alone analytically, as the number of operations exceeds what is possible to perform by any computer as of today. This challenge is often referred to as the  $\mathcal{N}$ -body problem [Klimontovich, 1967]. However, the level of accuracy available from a single particle description is unnecessary in most studies and therefore it is natural to introduce descriptions able to reduce the complexity. In the following sections, two such examples of plasma descriptions with lower complexity are given.

### 2.2.2 Kinetic description

Kinetic description, also known as the Plasma Kinetic theory, is one way of avoiding the  $\mathcal{N}$ -body problem, through the introduction of probability density functions

$$f(\mathbf{x}, \mathbf{v}, t) = \langle N(\mathbf{x}, \mathbf{v}, t) \rangle ,$$

where  $\langle \dots \rangle$  means to take the ensemble average over all possible realizations of the plasma consistent with some constraint, e.g. an ensemble of equal temperature plasma in thermal equilibrium. Assuming that the properties of  $f(\mathbf{x}, \mathbf{v}, t)$  are similar to that of a fluid (continuity, incompressibility, etc.), it is possible to describe the plasmas temporal evolution with Vlasov's equations

$$\frac{\partial f(\mathbf{x}, \mathbf{v}, t)}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{v}, t) + \frac{q}{m}(\mathbf{E}(\mathbf{x}, t) + \mathbf{v} \times \mathbf{B}(\mathbf{x}, t)) \cdot \nabla_{\mathbf{v}} f(\mathbf{x}, \mathbf{v}, t) = 0 , \quad (2.11)$$

---

<sup>2</sup> $\delta(\mathbf{r}) = 0$  for  $\mathbf{r} \neq \mathbf{0}$ ,  $\delta(\mathbf{0}) = \infty$  and  $\int \delta(\mathbf{r}) d\mathbf{r} = 1$ .

having only made the assumption of no collisions. For the probability density functions  $f(\mathbf{x}, \mathbf{v}, t)$  to represent a probabilistically acceptable system there are some requirements that need to be met:

- $f(\mathbf{x}, \mathbf{v}, t) \geq 0$  for all  $\mathbf{x}$ ,  $\mathbf{v}$  and  $t$ .
- Integral over all space and velocity space of  $f(\mathbf{x}, \mathbf{v}, t)$  equals the total number of particles ( $\int \int f(\mathbf{x}, \mathbf{v}, t) d\mathbf{x} d\mathbf{v} = \mathcal{N}$ )
- At the extreme velocity boundaries ( $v \rightarrow \pm\infty$ ),  $f(\mathbf{x}, \mathbf{v}, t)$  has to be zero.

Plasma Kinetic theory's strength is its ability to reduce the complexity of the problem without losing all detailed information of the problem. Its strongest limitation is by far the assumption of no collisions, which is a good approximation in most astro- and space-physics problems.

A steady state solution of Vlasov's equations is when there is (no longer) any temporal change ( $\partial/\partial t = 0$ ), i.e.

$$\mathbf{v} \cdot \nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{v}, t) + \frac{q}{m} (\mathbf{E}(\mathbf{x}, t) + \mathbf{v} \times \mathbf{B}(\mathbf{x}, t)) \cdot \nabla_{\mathbf{v}} f(\mathbf{x}, \mathbf{v}, t) = 0 . \quad (2.12)$$

This steady-state equation supports a large class of solutions [Bernstein et al., 1957], where the double layers discussed in the present thesis only represent one part.

### 2.2.3 Fluid description

As a further simplification of how to describe a plasma, a fluid model can be introduced. In this model, the only quantities kept are *bulk* parameters describing the plasma as an electromagnetic fluid. These quantities are the particle density, charge density, bulk velocity and current density, defined as:

$$\begin{aligned} n &= n(\mathbf{x}, t) = \sum_s n_s = \sum_s \int f_s(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} , \\ \rho_e &= \rho_e(\mathbf{x}, t) = \sum_s q_s n_s , = \sum_s q_s \int f_s(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} , \\ n\mathbf{u} &= n(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t) = \sum_s n_s \mathbf{u}_s = \sum_s \int \mathbf{v} f_s(\mathbf{x}, \mathbf{v}, t) d\mathbf{v} , \\ \mathbf{j} &= \mathbf{j}(\mathbf{x}, t) = \sum_s q_s n_s \mathbf{u}_s , \end{aligned}$$

where  $s$  represents the particle species involved.

By multiplying Vlasov's equations (2.11) with  $m$ ,  $m\mathbf{v}$  and  $m\mathbf{v}\mathbf{v}$  in turn, and then integrating over velocity space, one retrieves the fluid equations for conservation of mass, momentum and energy

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \quad (2.13)$$

$$\rho \left( \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) = \rho_e (\mathbf{E} + \mathbf{u} \times \mathbf{B}) - \nabla p \quad (2.14)$$

$$\frac{\partial}{\partial t} \left( \frac{\rho u^2}{2} + \frac{3p}{2} \right) + \nabla \cdot \left( \frac{\rho u^2 \mathbf{u}}{2} + \frac{3p\mathbf{u}}{2} \right) = \mathbf{j} \cdot \mathbf{E} - \nabla \cdot (\rho \mathbf{u}) . \quad (2.15)$$

Here  $\rho = nm$  is mass density,  $p$ ,  $\rho u^2/2$  and  $3p/2$  are the plasma pressure, kinetic energy and thermal energy, respectively, and  $\rho_e$  is the charge density. Combining equations

(2.13)-(2.15) with Maxwell's equations and an equation of state yields a complete set of equations, which can be solved for a given plasma.

Care must be taken with double layers when using a fluid description. To be able to obtain the non-zero charge density typical of double layers, the plasma needs to be separated into two fluids minimum, one fluid for each sign of charge. Note that in ideal magnetohydro-dynamics (MHD), which is the most frequently used fluid description in plasma physics, double layers can not exist. This is due to the assumption of infinitely high conductivity, hence electric fields can not be sustained and the dominant force in the plasma becomes the  $\mathbf{j} \times \mathbf{B}$ -force.

## 2.3 Basic quantities

In plasma physics, there are four quantities describing fundamental plasma processes; thermal velocity, Debye length, plasma frequency and the plasma parameter. In this section, these quantities are derived and their meaning interpreted.

### 2.3.1 Thermal velocity

Thermal velocity is the typical velocity of the thermal motion of the particles that make up a gas, liquid, etc. (in this case a plasma) and likewise a measure of temperature. When dealing with probability density functions, thermal velocity is a measure of the width of the function in velocity space. The definition for thermal velocity used in this thesis is

$$v_{ths} = \sqrt{\frac{\kappa T_s}{m_s}}, \quad (2.16)$$

where  $\kappa$  is Boltzmann's constant,  $T_s$  the temperature and  $m_s$  the mass of particle species  $s$ . More generally the thermal velocity is found from the relation  $\frac{1}{2}m\langle v^2 \rangle = \frac{3}{2}\kappa T$  for monatomic ideal gases with three degrees of freedom, where  $v_{th}^2 = \langle v^2 \rangle$ . The definition (2.16) ignores a numerical constant for convenience.

### 2.3.2 Debye length

Shielding of the potential originating from a point charge,  $q$ , occurs when a plasma acts as a cover for  $q$  by accumulating opposite charge density in the close vicinity (due to Coulomb interactions). The consequence of this will be that the plasma outside this cover no longer can see the potential of the charge  $q$ , but only the *collective behavior* of the cover and  $q$ . The Debye length is the scale length at which the potential originating from  $q$ , is shielded by the surrounding plasma with a factor of  $\exp(-1)$ . To find the exact form of this relationship a test charge of charge density similar to a  $\delta$ -function is introduced. Further, the assumption of spherical symmetry reduces this problem to one dimension ( $\mathbf{r} \rightarrow r = |\mathbf{r}|$ ). Inserting the above conditions into the electrostatic Poisson's equation (2.6) yields

$$\frac{d^2\phi}{dr^2} = \frac{e}{\epsilon_0} \left( n_e(r) - n_i(r) - \frac{q}{e}\delta(r) \right), \quad (2.17)$$

where  $e$  is the elementary charge,  $n_e$  and  $n_i$  particle densities of electrons and singly charged positive ions, respectively, and  $q\delta(r)$  the test charge. Ions are assumed immobile, i.e.  $n_i = n_0$ . Further we assume electrons to be in thermal equilibrium, hence have a centered time stationary Maxwellian probability density function under the influence of the electrostatic potential  $\phi$ . Integrating the probability density function over velocity space yields a Boltzmann distribution of the electron density

$$n_e(r) = n_0 \exp(e\phi(r)/\kappa T_e).$$



To be able to solve (2.17) analytically, the latter expression is approximated by its Taylor expansion, ignoring higher order terms, i.e.

$$n_e(r) \approx n_0 \left( 1 + \frac{e\phi(r)}{\kappa T_e} \right) \quad \text{for} \quad \frac{e\phi(r)}{\kappa T_e} \ll 1.$$

The solution of (2.17) then becomes

$$\phi(r) = \frac{q^2}{4\pi\epsilon_0 r} \exp\left(\frac{-r}{\lambda_D}\right),$$

where  $\lambda_D$  is a scale length of the decrease in potential, the Debye length;

$$\lambda_{De} = \sqrt{\frac{\epsilon_0 \kappa T}{e^2 n_0}}. \quad (2.18)$$

Similar derivations can be done for an arbitrary particle specie  $s$  with the result

$$\lambda_{Ds} = \sqrt{\frac{\epsilon_0 \kappa T_s}{e^2 n_0}}, \quad (2.19)$$

and combined into a common Debye length

$$\lambda_D = \left( \sum_s \frac{1}{\lambda_{Ds}} \right)^{-1}. \quad (2.20)$$

### 2.3.3 Plasma frequency

When a plasma is perturbed, an electric field  $\mathbf{E}$  will appear attempting to restore neutrality. Assuming that this perturbation is applied to a homogeneous, isotropic cold plasma (pressure terms can be ignored), ions are immobile, and the perturbations (subscript 1) are small compared to the background quantities (subscript 0),

$$\begin{aligned} n_e &= n_0 + n_1, \\ \mathbf{v}_e &= \mathbf{v}_0 + \mathbf{v}_1 = \mathbf{v}_1, \\ \mathbf{E} &= \mathbf{E}_0 + \mathbf{E}_1. \end{aligned}$$

Inserting the perturbed quantities into the equations of continuity and momentum for electrons

$$\begin{aligned} \frac{\partial n_e}{\partial t} + \nabla \cdot (n_e \mathbf{v}_e) &= 0, \\ mn_e \left( \frac{\partial \mathbf{v}_e}{\partial t} + (\mathbf{v}_e \cdot \nabla) \mathbf{v}_e \right) &= -en_e \mathbf{E}, \end{aligned}$$

and linearizing this, where higher order terms are neglected and solutions of the form  $\sim \exp(i(kx - \omega t))$  are assumed, one arrives at the dispersion relation

$$\omega^2 \equiv \omega_{pe}^2 = \frac{n_0 e^2}{\epsilon_0 m}. \quad (2.21)$$

where  $\omega_{pe}$  is the electron plasma frequency. Similar results can be calculated for an arbitrary particle specie  $s$ , giving

$$\omega_{ps} = \sqrt{\frac{(Ze)^2 n_0}{\epsilon_0 m_s}}, \quad (2.22)$$

where  $Z$  is the proton number. If  $s$  is an ion,  $Z = -1$  if  $s$  is an electron. Combining the plasma frequency of the different particle species, a common plasma frequency can be found

$$\omega_p^2 = \sum_s \omega_{ps}^2, \quad (2.23)$$

of which the entire plasma will oscillate. Taking the inverse of the plasma frequency gives an estimate of the characteristic time of temporal changes in the plasma. For convenience, a factor of  $2\pi$  has been ignored.

It is also worth noticing that the introduced quantities relate to one another as follows

$$\frac{v_{ths}}{\lambda_{Ds}\omega_{ps}} = \frac{\frac{\sqrt{\kappa T_s}}{m_s}}{\sqrt{\frac{e^2 n_s}{\epsilon_0 m_s}} \sqrt{\frac{\epsilon_0 \kappa T_s}{e^2 n_s}}} = \frac{\sqrt{\frac{\kappa T_s}{m_s}}}{\sqrt{\frac{\kappa T_s}{m_s}}} = 1.$$

### 2.3.4 The plasma parameter

The plasma parameter is a dimensionless quantity found through multiplying the particle density with the Debye length cubed,

$$N_p = n\lambda_D^3. \quad (2.24)$$

The plasma parameter provides a tool to decide whether a plasma is dominated by the particles' discrete nature or collective behaviour. As a first approximation, the interactions of charged particles can be assumed shielded by the Debye length, and that the dominant interaction under equilibrium conditions takes place between particles separated by a distances smaller than the Debye length. If the plasma parameter is small,  $N_p \sim 1$ , the discrete nature of the particle distribution is important (this is often the case in semiconductors, for instance). In the opposite limit,  $N_p \gg 1$  which is the case in most laboratory, astrophysical and space plasmas, a "smeared-out" representation of the particles will be sufficient. This limit is dealt with by Vlasov's equations [Chen, 1984; Pécseli, 2006].

## 2.4 Plasma physics through numerical simulations

The forces acting on the charged particles of a plasma (equation (2.1)) are non-linear and self-consistent and therefore too complex to be solved analytically. Numerical simulations provide a tool for calculating these forces by introducing fictional plasmas in computer programs, thus enabling full control and diagnostic abilities of the processes. A full scale model of the plasma is in essence equivalent to the single particle description discussed in section 2.2.1. As argued before, a single particle description is too complex to be solved, even numerically, hence numerical methods reducing the complexity need to be introduced.

The different numerical methods used to simulate plasmas can be divided into the same classifications as introduced in section 2.2; particle-, kinetic- and fluid description, where different methods are applicable to different problems. Particle and kinetic descriptions have proven successful in describing basic physical problems in which the probability distribution functions deviate significantly from a local Maxwellian distribution. Examples of such problems are wave-particle resonance, particle trapping, etc. Fluid description, usually in the form of magneto-hydro-dynamics (MHD), on the other hand, have often been applied to large scale systems directly related to the behaviour of experimental devices. A combination of particle- and MHD-codes, often referred to as *hybrid* codes, where fluid or particle descriptions apply to different components of a

given plasma, has become more frequent in recent years [Ledvina et al., 2008; Tang and Chan, 2005].

Below, three different numerical models, or schemes, are presented; the Particle-in-Cell method, Vlasov method and Fluid description. Each scheme is based on the plasma descriptions discussed in section 2.2. The Particle-in-Cell method combine particle and kinetic descriptions, while the Vlasov and Fluid methods are purely kinetic and fluid descriptions, respectively. All three schemes aim at solving the equations describing the plasma with as high *accuracy* and as low *complexity* and *number of force evaluations per timestep* as possible. Fulfilling all three criteria at the same time is not possible, hence compromises have to be made.

Complexity and accuracy of the schemes are given the notations  $\mathcal{O}(\mathcal{N}^n)$  and  $\mathcal{O}(\Delta^m)$ . Here  $\mathcal{N}$  is the particle number and  $\Delta$  is the resolution (spatial or temporal). The numbers  $n$  and  $m$  will vary between the schemes. Since present simulations are performed with a Particle-in-Cell code, this scheme is given extra attention and the other two are only mentioned by their overall characteristics.

Finally, a short discussion of boundary conditions often used in plasma physics simulations is given.

### 2.4.1 The Particle-in-Cell method

In the Particle-in-Cell method (PIC), the positions of each single particle are kept as continuous variables. The force acting upon the particles, however, is not calculated for each single particle, but rather on a fixed spatial grid of size  $\mathcal{N}_{cell} \ll \mathcal{N}$ . Prior to each force calculation, the charge of each particle is mapped down onto the spatial grid, through a weighting procedure, details of which will be discussed later.

With the grid containing information of the charge density, electrostatic potential and electric field is calculated, through Poisson's equation (2.6), from the charge density on the spatial grid. The electric field, converted to the force acting on each particle, is mapped back to the particles through a reversed weighting procedure. The mapping of particle charges down onto the spatial grid and back are procedures of complexity  $\mathcal{O}(\mathcal{N})$ , while solving the field equations for the spatial grid is of complexity  $\mathcal{O}(\mathcal{N}_{cell})$ . As long as  $\mathcal{N} \gg \mathcal{N}_{cell}$ , the complexity of the PIC method equals  $\mathcal{O}(\mathcal{N}) + \mathcal{O}(\mathcal{N}_{cell}) \approx \mathcal{O}(\mathcal{N})$ , which is a vast improvement to the original complexity of  $\mathcal{O}(\mathcal{N}^2)$ .

Within the PIC method, there are several sub-methods with different approaches to how the mapping between particles and spatial grid is performed. Two of these sub-methods are discussed here; the *Nearest Grid Point* and *Cloud-in-Cell* methods. Both methods are presented in a one dimensional model. However, the described procedures and introduced stability requirements can easily be generalized to higher dimensions.

The Nearest Grid Point (NGP) method finds for each particle the grid point closest to the particles position, and as the name implies, assigns the charge of the particle to that grid point. This method is the simplest of the PIC methods, hereby the name *the zeroth order PIC method*. Although the method is fast, noise is a problem when particles transit between cells. As the boundaries between two neighbouring cells are discrete, the charge density in both cells can change abruptly when a particle transits between cells, giving rise to fields with no foundation in real physics. The Cloud-in-Cell (CIC) method avoids this problem by describing particles as triangular clouds, two cells wide, allowing the particle to distribute its charge to not only one cell, but two, smoothing the transition of particles between cells. CIC is able to sustain a lower noise level without increasing the complexity  $\mathcal{O}(\mathcal{N}) + \mathcal{O}(\mathcal{N}_{cell}) \approx \mathcal{O}(\mathcal{N})$ . The CIC method was chosen in the present implementation of the PIC method.

The cloud shape is illustrated in figure 2.3. The functional value of the triangle, which represents the weighting of the cloud, is given at the vertical axis. The charge of the

particle is assigned to each of the two cells the triangle covers according to the weight of the triangle at each grid point,  $w_i = 1 - (x - x_i)/(x_{i+1} - x_i)$  and  $w_{i+1} = 1 - w_i$ . Written with equations, for a particle with position  $x \in [x_i, x_{i+1}]$  where  $x_i$  is the position of grid point  $i$ , the weighting procedure is

$$Q_i = qw_i = q \left( 1 - \frac{x - x_i}{x_{i+1} - x_i} \right), \quad Q_{i+1} = qw_{i+1} = q \frac{x - x_i}{x_{i+1} - x_i},$$

where  $Q_i$  is the charge assigned to each grid point  $i$  of the particle's charge  $q$ .

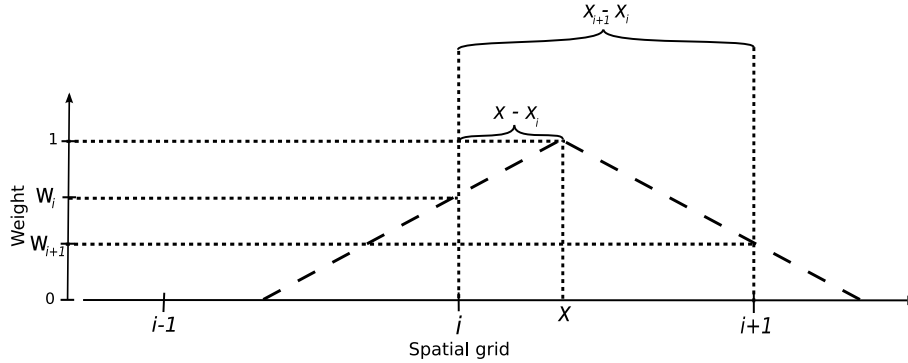


Figure 2.3: Illustration shows an example of how a particle in position  $x$  is weighted down to spatial grid points with positions  $x_i$  and  $x_{i+1}$ . The height of the triangular shaped particle represents the weighting of the particle.

Accuracy and stability requirements for this method are as follows:

- The size of the cell must be large enough to contain "many particles", typically  $> 10^3$  to reduce noise. If the number of particles is lower, the signal to noise ratio of the PIC method will approach unity [Birdsall and Langdon, 1985].
- The spatial variation  $\Delta P$  of a field  $P$  (e.g. charge density, electric field or electrostatic potential) within a cell must satisfy the condition  $\Delta P \ll P$  [Birdsall and Langdon, 1985].
- During one time step the movement of any particle must be less than the size of one cell ( $v \cdot \Delta t < \Delta x$  where  $\Delta x$  and  $\Delta t$  is the spatial grid size and temporal integration step length), a condition also referred to as the *Courant, Friedrich and Lewy condition* [Courant et al., 1928].

## 2.4.2 The Vlasov method

The Vlasov method treats phase space as a continuum and solves the system by integrating Vlasov's equations (2.11). This approach avoids the statistical errors often encountered in particle simulations, and in that respect this method has proven successful. This approach is not as adaptable as a single particle description (e.g PIC), especially in multidimensional problems [Birdsall and Langdon, 1985].

## 2.4.3 Fluid description

A numerical fluid description of a plasma is equal to computational fluid dynamics, with the added effects of electric and magnetic fields, and their coupling to the plasma. The

equations left to solve are as described in section 2.2.3, given the bulk properties of the plasma (density, bulk velocity, current density, etc.). To assure that the solution is physically acceptable the divergence of the magnetic field must be zero ( $\nabla \cdot \mathbf{B} = 0$ ) throughout the simulations.

#### 2.4.4 Boundary conditions

Boundary conditions can be used to represent the surrounding conditions when studying a physical system of finite size. In the context of plasma physics, such conditions can represent e.g. fixed potentials or constant currents of an experimental device. In nature they can represent e.g. a plasma in contact with a medium of fixed potential or high inductance or the surrounding plasma, represented by fixed potential or constant current boundary conditions, respectively. It has further been proven that electronic circuits can represent plasmas in nature [Smith, 1982]. Therefore, if boundary conditions representing the electronic circuit are known, they may also apply for the represented plasma.

In classical "lumped electronic circuits," two basic generators are distinguished: fixed voltage and fixed current generators. Realistic generators are built out of these by adding models for internal impedances [Horowitz and Hill, 1980]. The fixed voltage generator is one maintaining a constant potential difference between its two terminals, irrespective of the load. Similarly, the fixed current generator maintains a constant current, again irrespective of the load. Both generators represent idealized, extreme cases: the constant voltage generator will support an infinite current when short-circuited, while the constant current generator will force a current even in case of infinite load, which will imply an infinite voltage across the terminals.

The mathematical equivalents to boundary conditions used in the present studies are the types Dirichlet and von Neumann. Dirichlet keeps the value of the potential fixed at a known value, while von Neumann keeps the derivative of the potential (i.e. the electric field) fixed. Thus combining equal type boundary conditions to both boundaries for each of the spatial dimensions, the combinations Dirichlet-Dirichlet and von Neumann-von Neumann represent the constant voltage and current generators discussed above, respectively. The boundary condition combinations mixing von Neumann and Dirichlet can be considered as hybrids of these. What kinds of physical properties these combination may resemble is not known to the author at the present time.



## Chapter 3

# The Physics of Double Layers

A double layer (DL) is a structure consisting of two equal but oppositely charged layers. The present study considers DLs in plasma typical of those found in astrophysical and space plasmas. Within the DL, the charge density, potential and electric field will vary as illustrated in figure 1.1. For the structure to be "just" a DL, the electrostatic potential is monotonically increasing/decreasing (i.e.  $d\phi/dx \geq 0$  or  $d\phi/dx \leq 0$  for all  $x$ ). If, on the other hand, the potential structure has local minima (ion holes) and maxima (electron holes) within its boundaries, strictly speaking, it consists of a DL *and* ion-/electron holes.

Some typical DL properties often found in literature [Block, 1978]:

- The net potential difference  $\phi_{DL}$  obeys the relation

$$\phi_{DL} \gtrsim \frac{\kappa T}{e}$$

where  $T$  is the temperature of the coldest plasma bordering to the DL.

- Quasi neutrality is locally violated in both space charge layers, while the overall charge neutrality ( $\int_0^\ell \rho(x)dx \approx 0$ ) and charge neutrality at the boundaries ( $\rho(x=0) = \rho(x=\ell) = 0$ ) is conserved (given the DL defined within  $x \in [0, \ell]$ ).
- The DL is several Debye lengths thick.

In addition, a typical but not necessary condition is that the collisional mean free path is much longer than the DL thickness. Experimental as well as theoretical evidence indicate that as long as collisions play an appreciable role, a DL will not be formed [Block, 1978].

DLs are in general divided into two classes, strong and weak, dependent on how their potential energy  $e\phi_{DL}$  compares to the plasma thermal energy  $\kappa T$ , i.e.

$$\begin{aligned} e\phi_{DL} &\gg \kappa T \Rightarrow \text{Strong DL} \\ e\phi_{DL} &\lesssim \kappa T \Rightarrow \text{Weak DL.} \end{aligned}$$

In this chapter, an overview of the occurrence of DLs in nature is given. Particle and DL interactions are accounted for, and the typical properties and basic theories describing a DL are presented. Strong and weak DLs are then thoroughly discussed before a one dimensional theoretical model of DLs is introduced based on the BGK-solutions and water-bag model. Finally, previous and present numerical models of DLs are presented at the end of this chapter.

### 3.1 Double layers in nature and experiments

The first known work on DLs was done by Langmuir in 1929 in his article titled "*The Interaction of Electron and Positive Ion Space Charges in Cathode Sheaths*" [Langmuir, 1929]<sup>1</sup>. It was not until the 1950s that a detailed study of DL was performed in a laboratory [Schönhuber, 1958]. In the same decade, Hannes Alfvén suggested the existence of a DL in connection with the aurora, responsible for accelerating particles from the magnetosphere into the ionosphere [Alfvén, 1958].

Since then, theoretical, experimental and numerical studies have been done on DLs. Early investigations done on magneto- and ionospheric physics were the experiments of [McIlwain, 1960] and [Mozer et al., 1977] who, with their rocket and satellite, respectively, could measure the existence of a DL above the polar regions. The most definite proof of a DL came with the Viking satellite operating in the years 1986 and 1987. The Viking satellite was equipped with two 40m long booms measuring the potential difference between them. With this instrument, first hand evidence was given of a potential profile, typical to that of a DL, in our magnetosphere. In magnetospheric physics, the term DL is only used when the electric field associated with the DL is parallel to the background magnetic field. Otherwise, they are referred to as *Electrostatic Shocks*.

DLs are thought to be responsible for accelerating particles from high altitudes (typically magnetospheric altitudes) down into the auroral region (ionosphere) enhancing the Aurora. This has been confirmed by numerous experiments, e.g. [McIlwain, 1960; Mozer et al., 1977].

The polarity of the magnetospheric DL is generally so that electrons are accelerated downwards (earthwards), while positive ions upwards. For electrons to create the aurora, an electron energy higher than 1keV is required (typically what is observed in the ionosphere is 10 – 100keV). This energy must be supplied somewhere in the near Earth space, which is where the DLs enter the picture. The DL electric field has a typical strength of  $10^{-2}\text{V/m}$  giving a net potential change over the entire DL's width of several kV [Raadu, 1989], sufficient to fuel the high energy of auroral electrons. Simultaneously,  $O^+$  ions originating from the ionosphere are observed to be accelerated upwards with the same amount of energy as the electrons, leaving little doubt of the DL electric field's presence.

In the book on the Freja satellite [André, 1993] six different theories explaining a field aligned electric field in the magnetosphere are listed. Amongst these are DL, magnetic mirroring, collisionless thermoelectric fields, anomalous resistivity, Alfvén waves and lower hybrid waves. What all these theories have in common is that they predict the presence of the observed potential leaps in the magnetosphere, however, they argue for six different creation mechanisms. More recent studies have concluded that DL is the correct model of these six [Ergun et al., 2003], and is as of today a widely accepted model.

### 3.2 Particle interactions

The particle-DL interaction depends on the particles' kinetic energy, direction of motion and position relative to the DL's position and potential  $\phi_{DL}$ . The interactions allows for particles to be categorized in the following ways (name of categories written in *italic*):

1. Any particle with charge  $q < 0$  at the lower potential boundary of the DL, or  $q > 0$  at the high potential boundary, moving towards the DL will be *accelerated* through, irrespective of the particle's energy.

---

<sup>1</sup>Langmuir called DLs Double Sheaths.



2. Any particle with charge  $q < 0$  at the higher potential boundary of the DL, or  $q > 0$  at the lower potential boundary, moving towards the DL with kinetic energy  $E_k > e\phi_{DL}$  will be *decelerated* but still able to pass through the DL.
3. As described in the latter case, but now with kinetic energy  $E_k \leq e\phi_{DL}$  will be *reflected* by the DL.

Throughout this thesis the terms. *Free* and *trapped* particles are used, *free* refers to particle population 1 and 2 while *trapped* are referring to particle population 3 in the list above.

Bernstein, Greene and Kruskal [Bernstein et al., 1957] formed a set of stationary, non-linear solutions for Vlasov's equations, referred to as BGK-solutions, where DLs can be recognized as one of these solutions. In this, they proved that in order to have BGK-solutions, hence also DLs, trapped particle populations<sup>2</sup> must be present. BGK-solutions will be further discussed in section 3.5.1.

### 3.3 Properties and related phenomena

Through experimental and theoretical studies of DLs, three properties which defines stable DLs have been found. These criteria are given the names scaling law, Bohm criterion and Langmuir condition. The scaling law gives an interpretation of previous numerical results to which combinations of DL potential leap and width form stable double layers. The Bohm criterion introduces a least allowed entry velocity of accelerated particles and the Langmuir condition introduces a relationship between the current densities of the two involved particle species. These DL properties are in the following presented and discussed.

Two well established phenomena in plasma physics, the Buneman instability and adiabatic cooling, have been found to play a significant role in DLs. The Buneman instability is a two-stream instability thought to be able to saturate into a DL in a current dominated plasma, while adiabatic cooling is a cooling process observed for the accelerated particles in the present simulations. These two phenomena are derived and their influence on DLs are accounted for.

#### 3.3.1 The scaling law

Previous simulations on DLs [Smith, 1982] argue for the presence of a scaling law, providing a relationship between the width and the strength of the DL. The scaling law predicts that

$$\phi_{DL} = \eta L^2, \quad (3.1)$$

where  $\phi_{DL}$  is given in units of  $\kappa T/e$  and  $L \equiv \ell/\lambda_{De}$ . Equation 3.1 is interpreted such that the DLs are stable if  $\phi_{DL} < \eta L^2$ . The value of  $\eta$  is argued for in the following way:

"Varying definitions may be used for  $L$ , but general agreement exists that  $\eta$  is of order 0.1. All strong 1-D double layers obey this scaling." [Smith, 1982]

The value of  $\eta$  and the conclusions found in [Smith, 1982] are based on results from several previous numerical studies of DLs. These studies will be further discussed in section 3.6. What all previous studies have in common is their support of a scaling law being present, that DLs with  $\phi_{DL} < \eta L^2$  are observed to be unstable.

<sup>2</sup>Trapped in the sense of having restricted movement due to a potential barrier.

### 3.3.2 The Bohm criterion

The study of [Block, 1978] argues for the presence of the DL space charge due to the natural density variations of the particle populations involved, stating that

$$n_i \sim \exp\left(-\frac{e\phi}{\kappa T_{ti}}\right),$$

$$n_e \sim u_e^{-1} = \left(u_{De0}^2 + \frac{2e\phi}{m_e}\right)^{-1/2},$$

are valid at the low potential boundary, where the index  $ti$  indicates *trapped ions* and quantities  $u_e$  and  $u_{e0}$  are the drift velocities of free electrons at the boundary and inward respectively. The densities are equal at the boundary, but just inside of it, the trapped ion density appear to drop more rapidly, if and only if  $m_e u_{e0}^2 > \kappa T_{ti}$ . This model has not taken the free electron temperature into consideration. However doing so using fluid theory, [Block, 1972] found the minimum drift velocity of the electrons to be given by

$$m_e u_{e0}^2 = \kappa(\gamma T_{fe} + T_{ti}), \quad (3.2)$$

where  $\gamma$  is the adiabatic constant of free electrons.

DLs are observed to be stable if and only if the minimum drift velocity (3.2) is sustained, a criterion known as the *Bohm criterion*<sup>3</sup>. Further studies, both experimentally and numerically have given similar results for the Bohm criterion, summarized in [Smith, 1982], to be given as

$$u_e^2 \geq \left(\gamma_e + \frac{T_i}{T_e}\right) v_{the}^2, \quad (3.3)$$

$$u_i^2 \geq \left(\gamma_i + \frac{T_e}{T_i}\right) v_{thi}^2, \quad (3.4)$$

for non-relativistic DLs. In the asymptotic limit of weak DLs, the Bohm criterion simplifies to

$$u_s^2 \geq \frac{\gamma_s T_s}{m_s}. \quad (3.5)$$

### 3.3.3 The Langmuir condition

Through theoretical studies (analytic as well as numerical) a relationship between the current densities of electrons and ions in a strong DL has been obtained. In a one dimensional model, the Langmuir condition can be derived from Poisson's equation (2.6)

$$-\epsilon_0 \frac{d^2 \phi}{dx^2} = en_i(\phi) - en_e(\phi),$$

conservation of flux

$$\frac{d}{d\phi}(n(\phi)v(\phi)) = 0 \quad \Rightarrow \quad \begin{cases} -en_e(\phi)v_e(\phi) = j_e \\ en_i(\phi)v_i(\phi) = j_i \end{cases},$$

and the equations for the energy of singly charged particles under the influence of a DL potential  $\phi$

$$\frac{1}{2}m_e v_e^2(\phi_0) - e\phi_0 = \frac{1}{2}m_e v_e^2(\phi) - e\phi,$$

$$\frac{1}{2}m_i v_i^2(\phi_{DL}) + e\phi_{DL} = \frac{1}{2}m_i v_i^2(\phi) + e\phi.$$

---

<sup>3</sup>A similar criterion was derived for wall sheaths by Bohm [1949], hence the name.

Here, electrons are assumed to enter at the low potential boundary, where  $\phi = \phi_0$ , and ions at the high potential boundary, where  $\phi = \phi_{DL}$ .

Expressing the  $en(\phi)$  terms with  $j/v(\phi)$  retrieved from the flux conservation and  $v(\phi)$  with the energy equations, where the negative solution is chosen when taking the root for the ion energy, yields

$$\begin{aligned}
-\epsilon_0 \frac{d^2 \phi}{dx^2} &= -\frac{j_i}{\sqrt{\frac{2e}{m_i}(\phi_{DL} - \phi)}} + \frac{j_e}{\sqrt{\frac{2e}{m_e} \phi}} \\
&\Downarrow \\
-\epsilon_0 \frac{d\phi}{dx} \frac{d^2 \phi}{dx^2} &= -\frac{\epsilon_0}{2} \frac{d}{dx} \left( \frac{d\phi}{dx} \right)^2 \\
&= -j_i \frac{\frac{d\phi}{dx}}{\sqrt{\frac{2e}{m_i}(\phi_{DL} - \phi)}} + j_e \frac{\frac{d\phi}{dx}}{\sqrt{\frac{2e}{m_e} \phi}} \\
&= 2j_i \sqrt{\frac{m_i}{2e}} \frac{d}{dx} \sqrt{\phi_{DL} - \phi} + 2j_e \sqrt{\frac{m_e}{2e}} \frac{d}{dx} \sqrt{\phi} \\
&\Downarrow \\
-\frac{\epsilon_0}{2} \left( \frac{d\phi}{dx} \right)^2 + W &= 2j_i \sqrt{\frac{m_i}{2e}} \sqrt{\phi_{DL} - \phi} + 2j_e \sqrt{\frac{m_e}{2e}} \sqrt{\phi},
\end{aligned}$$

where  $W$  is a constant of integration. Just outside the DL on both sides, where  $\frac{d\phi}{dx} = 0$ ,  $W$  becomes

$$W = 2j_i \sqrt{\frac{m_i}{2e} \phi_{DL}} \quad \text{for } \phi = \phi_{DL}$$

and

$$W = 2j_e \sqrt{\frac{m_e}{2e} \phi_{DL}} \quad \text{for } \phi = 0.$$

Combining these by eliminating  $W$  lead to the Langmuir condition, i.e. the ratio of the current densities

$$\frac{j_e}{j_i} = \sqrt{\frac{m_i}{m_e}}. \quad (3.6)$$

The Langmuir condition has been confirmed by numerical studies of strong DLs, where DLs fulfilling the Langmuir condition are found to be stable, otherwise *unstable*. Similar results has not been found for weak DLs, and it is therefore concluded that the Langmuir condition applies to strong DLs only.

### 3.3.4 Buneman instability

The Buneman instability is thought to be the dominant plasma instability in relation to DLs and responsible for creating strong DLs. In magnetospheric physics, the Buneman instability arise along the magnetic field line, when thermal electrons are accelerated by an external electric field along the ambient magnetic field. Such electric fields are induced when the magnetosphere undergoes dynamic changes of its structure such as magnetic reconnection or reconfiguration [Raadu, 1989]. Under the electric field imposed along the magnetic field, the thermal electrons are accelerated to a finite drift velocity  $u_e$  relative to the thermal ions.

When  $u_e$  exceeds a critical value, the Buneman instability sets in to convert the drift kinetic energy to electrostatic field energy and thermal energies of electrons and ions

[Omura et al., 2008]. The instability is dominated by the fastest growing mode and its harmonics.

In the following, the basic properties of the Buneman instability are derived for probability density functions associated with the DLs. As the present simulations emphasize strong double layers, adiabatic cooling (to be discussed in section 3.3.5) of the particles makes finite temperature effects negligible. Therefore a simplifying assumption using a cold plasma model can be made for ions as well as electrons [Pécse, 2006].

Consider the linearized continuity equation of the form

$$\frac{\partial}{\partial t}n_s + \frac{\partial}{\partial x}(u_s n_s) = 0,$$

for each of the species  $s$ , restricting the analysis to one spatial dimension along the  $x$ -axis. The corresponding momentum equations for cold plasmas are

$$\frac{\partial}{\partial t}u_s + u_s \frac{\partial}{\partial x}u_s = \frac{q_s}{m_s} E,$$

where  $q_s$  and  $m_s$  denote the particle charge and mass, respectively. The particle density does not enter the momentum equation since it cancels on both sides of the expression in the present cold-plasma limit. The set of equations is completed by Poisson's equation

$$\frac{\partial}{\partial x}E = \frac{1}{\epsilon_0} \sum_s q_s n_s.$$

These basic equations are subsequently linearized. Assume a frame of reference where one of the species is at rest and the others can be moving with some known velocity. To find a dispersion relation  $\omega = \omega(k)$ , a plane wave approximation is used, i.e., it is assumed that all space-time variations are of the form  $\sim \exp(-i(\omega t - kx))$ .

For the low potential side of a DL there are in general three populations, one ion and an accelerated and decelerated electron population. Similarly the populations on the high potential side are one electron and an accelerated and decelerated ion population. The basic equations for electrostatic waves in a one dimensional case for ions at rest and two electron components (as for the low potential side of the DL) moving with velocities  $U_{01}$  and  $U_{02}$ , and unperturbed densities  $n_{01}$  and  $n_{02}$  respectively, are linearized to give

$$\frac{\partial}{\partial t}\tilde{n}_s + U_{0s} \frac{\partial}{\partial x}\tilde{n}_s + n_{0s} \frac{\partial}{\partial x}\tilde{u}_s = 0$$

and

$$\frac{\partial}{\partial t}\tilde{u}_s + U_{0s} \frac{\partial}{\partial x}\tilde{u}_s = \frac{q_s}{m_s} \tilde{E}$$

with  $\tilde{\phantom{x}}$  indicating perturbed quantities. Poisson's equation is linear from the outset.

The dispersion relation for electrostatic waves in a one dimensional case for the present problem is found as

$$1 = \omega_{pe}^2 \left( \frac{m_e/m_i}{\omega^2} + \frac{n_{01}/n_0}{(\omega - kU_{01})^2} + \frac{n_{02}/n_0}{(\omega - kU_{02})^2} \right), \quad (3.7)$$

where  $\omega_{pe}$  is the plasma frequency obtained for the sum of the densities of the two components,  $n_0 \equiv n_{01} + n_{02}$ . Because of the overall charge neutrality, this density has to equal the ion density  $n_i$ . Equation (3.7) is somewhat problematic by containing several parameters  $U_{01}$ ,  $U_{02}$ ,  $n_{01}$  and  $n_{02}$ , so it is an advantage to discuss a simplified model.

To illustrate the properties of (3.7) the problem is reduced to the case with only *two* species (although this is not directly relevant for the double layer problem, but

the analysis becomes simpler and still illustrates the basic phenomena). For discussing this relation, it is an advantage to introduce the normalized variables  $\Omega \equiv \omega/\omega_{pe}$  and  $K \equiv kU_0/\omega_{pe}$ . We then have

$$1 = \frac{m_e/m_i}{\Omega^2} + \frac{1}{(\Omega - K)^2}. \quad (3.8)$$

This relation can be rewritten as a fourth order polynomial, and will have four roots, where some might be double roots. Since all coefficients are real, complex roots will appear in complex conjugate pairs, i.e. if we have one complex solution, also its complex conjugate will be a solution.

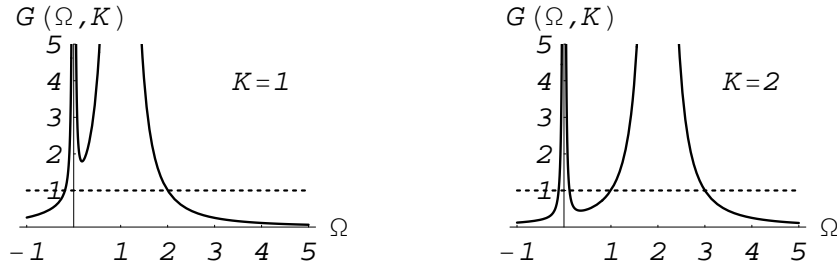


Figure 3.1: *Diagram for analyzing stability conditions for the Buneman instability. The figure assumes an artificial mass ratio of  $m_i/m_e = 100$ ; otherwise the curve would be extremely narrow in the vicinity of  $\Omega = 0$ . The figure to the left, for  $K = 1$ , corresponds to unstable conditions, the one with  $K = 2$  to stable conditions.*

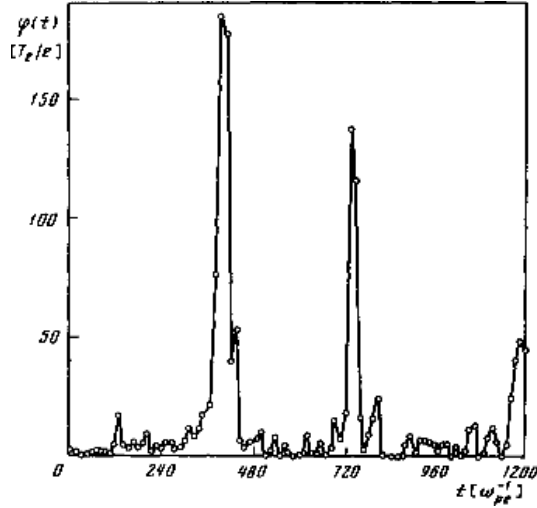


Figure 3.2: *The simulations of [Belova et al., 1980] proved how the Buneman instability can create transient DLs. The finite potential leap  $\phi_{DL}$  is plotted as a function of simulation time. Courtesy of figure: [Belova et al., 1980].*

At this point, it is practical to introduce a new function

$$G(\Omega, K) \equiv \left( \frac{m_e/m_i}{\Omega^2} + \frac{1}{(\Omega - K)^2} \right).$$

Since  $m_e/m_i$  is generally a small number,  $G(\Omega, K) \approx 1$  for  $\Omega - K = \pm 1$ , unless  $\Omega$  is close to zero. In which case also the term with coefficient  $m_e/m_i$  becomes important. To analyze the stability conditions of (3.8), it is best to use a graphic representation as the one in figure 3.1. Since  $G(\Omega, K) \rightarrow 0$  for  $\Omega \rightarrow \pm\infty$  and  $G(\Omega, K) \rightarrow \infty$  for  $\Omega \rightarrow 0$  and  $\Omega \rightarrow K$ , there must be at least two real solutions for  $G(\Omega, K) = 1$ . If the line  $G(\Omega, K) = 1$  is crossing the curve *four* places there are four real solutions, corresponding to stability. If we have only *two* crossings, there must be two more solutions, these being complex conjugate. One of these solutions then correspond to *instability*. It is evident from figure 3.1, that small wave numbers,  $K$ , are those giving unstable conditions.

The condition of marginal stability, with one double root at the coordinate where  $\partial G(\Omega, K)/\partial \Omega = 0$ , is readily found to be  $\Omega = K(m_e/m_i)^{1/3}/(1 + (m_e/m_i)^{1/3})$ , which inserted into the marginal stability condition  $G(\Omega, K) = 1$  gives

$$K_c = \sqrt{1 + 3(m_e/m_i)^{1/3} + 3(m_e/m_i)^{2/3} + m_e/m_i} \approx 1.$$

Note that even though  $m_e/m_i \ll 1$ , then  $(m_e/m_i)^{1/3}$  might be a nontrivial correction in  $K_c$ .

From the above given discussion, it is shown that waves with wave numbers  $K < K_c$  are unstable. It is possible to generalize this study by introducing more particle species/populations. However, the simplified model discussed here proves that generally long wavelengths, i.e. small  $K$ , are expected to be unstable, and therefore these results are representative also for more generalized cases.

The present discussion of the Buneman instability depends explicitly on the assumption of vanishing temperatures. A more general analysis of stability will be based on the Penrose criterion [Pécsele, 2006]. It is emphasized that a stability analysis as the one discussed here assumes a homogeneous plasma. A plasma of *finite* extent can be stable if all unstable wavelengths are longer than the extent of the plasma.

Previous studies [Belova et al., 1980; Smith, 1982] refer to simulations where the Buneman instability creates a DL when drift velocities are typically  $u_e \approx 1.8\sqrt{kT_e/m_i}$ . In figure 3.2, results from the numerical simulations of [Belova et al., 1980] show how the Buneman instability can result in a DL, through plotting the potential leap  $\phi_{DL}$  as a function of time. The DLs observed in the results from [Belova et al., 1980] are of transient nature, i.e. not time stationary. It is not known to what extent this might be the result of the boundary conditions used by these authors.

### 3.3.5 Adiabatic cooling

An inherent feature (often called "adiabatic cooling" [Pécsele, 2006]) of particle acceleration will be noticeable several places in the present simulations, and deserves to be mentioned specially. Ions and electrons are *accelerated* through the space charge region, and are lost at the end of the system. In order to obtain an expression for the ion distribution function, e.g. in the plasma, a current density is assumed present. This current density is associated with ions in the velocity interval  $(v, v + dv)$  at the incoming surface to be

$$dI_{i1} = n_{i0}ev\sqrt{\frac{m_i}{2\pi\kappa T}} \exp\left(-\frac{m_iv^2}{2\kappa T}\right) dv$$

where the temperature is taken to be that of the injected particles. The local ion density is  $n_{i0}$ . The injected particles are assumed to have a Maxwellian distribution. After acceleration through the potential leap, an ion has an energy  $\frac{1}{2}m_iu^2 = \frac{1}{2}m_iv^2 + e\Delta\phi$ , for  $u > 0$ .

In the plasma, these very same ions give a current contribution

$$dI_{i2} = n_{i0} e u \sqrt{\frac{m_i}{2\pi\kappa T}} \exp\left(-\frac{m_i u^2}{2\kappa T}\right) \exp\left(\frac{e\Delta\phi}{\kappa T}\right) du$$

since  $u du = v dv$ . Assume that the ion velocity distribution in the plasma is  $g(u)$ , normalized to unity,  $\int_{-\infty}^{\infty} g(u) du = 1$ , with the ion density being  $n_i = n_p$ . For steady state conditions, the ion current is the same through any cross section of the plasma in the direction transverse to the simulation direction. Therefore the following can be written  $dI_{i2} = n_p e u g(u) du$ , and find

$$n_{i0} \sqrt{\frac{m_i}{2\pi\kappa T}} \exp\left(-\frac{m_i u^2}{2\kappa T}\right) \exp\left(\frac{e\Delta\phi}{\kappa T}\right) = n_p g(u).$$

It would be tempting here to argue that  $n_p = n_{i0} \exp(e\Delta\phi/\kappa T)$ , but this would be in error. The problem is that only ions with velocities  $u > \sqrt{2e\Delta\phi/m_i} \equiv u_{im_i}$  are present in the plasma column, outside the potential leap. We have required  $g(u)$  to be normalized, but have

$$\sqrt{\frac{m_i}{2\pi\kappa T}} \int_{u_{im_i}}^{\infty} \exp\left(-\frac{m_i u^2}{2\kappa T}\right) du \neq 1,$$

so the normalization of  $g(u)$  must be ensured in a more subtle manner. To have the correct normalization, the requirement

$$n_p = n_{i0} \sqrt{\frac{m_i}{2\pi\kappa T}} \exp\left(\frac{e\Delta\phi}{\kappa T}\right) \int_{u_{im_i}}^{\infty} \exp\left(-\frac{m_i u^2}{2\kappa T}\right) du$$

is necessary, giving

$$g(u) = \frac{1}{1 - \operatorname{erf}\sqrt{e\Delta\phi/\kappa T}} \sqrt{\frac{2m_i}{\pi\kappa T}} \exp\left(-\frac{m_i u^2}{2\kappa T}\right), \quad \text{for } u \geq \sqrt{\frac{2e\Delta\phi}{m_i}} \quad (3.9)$$

and  $g(u) = 0$  for  $u < \sqrt{2e\Delta\phi/m_i}$ . The error function  $\operatorname{erf}$  is defined

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt. \quad (3.10)$$

Figure 3.3 shows the variation of the ion distribution function for varying normalized plasma potential leaps,  $e\Delta\phi/\kappa T = 0, 0.5, 1.5$  and  $2.5$ . The electrons and ions are in thermal equilibrium at the boundary.

Note that the average velocity  $\langle u \rangle = \int u g(u) du$  increases with  $e\Delta\phi/\kappa T$ , thus calculating

$$\langle u \rangle = \frac{\exp(-e\Delta\phi/\kappa T)}{1 - \operatorname{erf}\left(\sqrt{e\Delta\phi/\kappa T}\right)} \sqrt{\frac{2\kappa T}{\pi m_i}},$$

as shown in figure 3.4.

Finally, an “effective temperature” for the ion or electron distribution is obtained as  $T_{\text{eff}} \equiv (m_i/\kappa) \langle (u - \langle u \rangle)^2 \rangle = \int (u - \langle u \rangle)^2 g(u) du$ . Also this result has an analytical expression, but it is rather lengthy, and need not be reproduced here. The result is shown graphically in figure 3.5. The effective “cooling” due to the bulk ion (or electron) acceleration over the potential leap  $\Delta\phi$  is sometimes called “adiabatic cooling”, and is generic to similar phenomena also in a wider context. It is important to note that the cooling only affects the parallel part of the velocity distribution function. The perpendicular part has the same Maxwellian distribution as at the injection, and the velocity distribution in a fully two or three dimensional system is therefore strongly anisotropic.

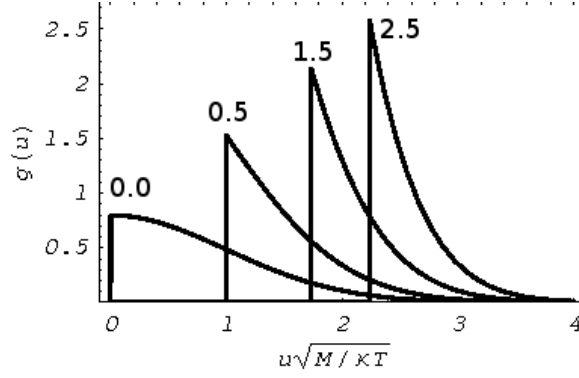


Figure 3.3: Variation of the ion distribution function with normalized plasma potential. It is visually apparent how the width of the distribution decreases with increasing potential leap,  $e\Delta\phi/\kappa T = 0, 0.5, 1.5$  and  $2.5$ .

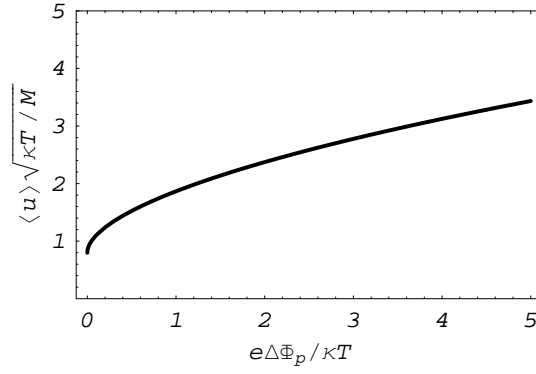


Figure 3.4: Variation of the normalized drift velocity with normalized plasma potential leap.

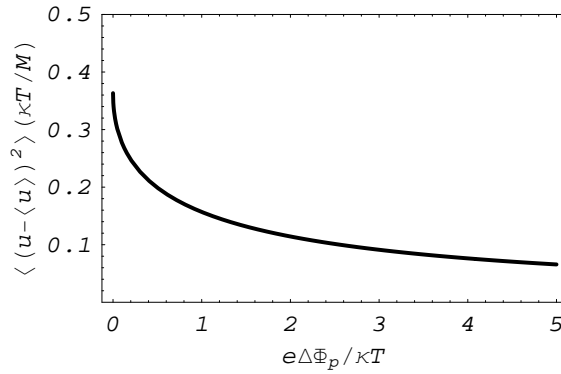


Figure 3.5: Variation of the normalized effective temperature of the distribution function with normalized plasma potential leap.



### 3.4 A summary of characteristic

Figure 3.6 illustrates three different types of DLs, represented with potential profiles (first row), phase space diagrams for ions and electrons (second and third row) and density profiles (fourth row). The first column represents a strong DL while the two other columns both represent weak DLs, subdivided into Slow Electron Acoustic and Slow Ion Acoustic DLs.

In the following, the properties illustrated in figure 3.6 are discussed and their relationship with the properties and phenomena discussed in the previous sections is given.

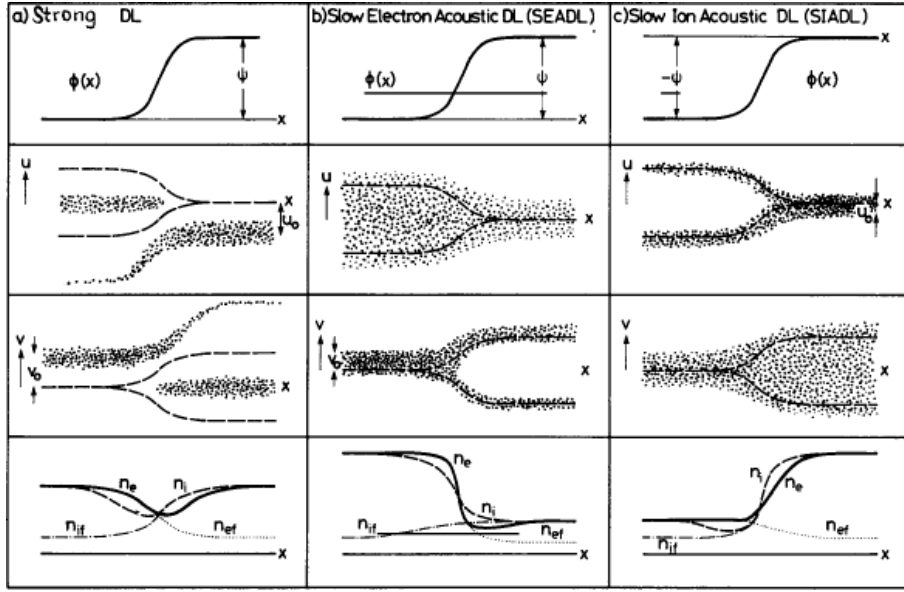


Figure 3.6: The three columns contain typical illustrations of strong, slow electron acoustic and slow ion acoustic DL. The rows contain for each column plots of the potential profile, the ion phase space diagram, the electron phase space diagram and density profiles for both species (density of the free populations and total for each specie). Adapting the notation of the present thesis to this figure means that  $\phi_{DL} = \psi$ ,  $u$  and  $v$  are the ion and electron velocities, respectively. Courtesy of figure: [Schamel, 1986].

#### 3.4.1 Strong double layers

Strong DLs can be characterized by these four properties, further explained in the text below:

- There are four populations of particles present (accelerated/reflected, ions/electrons).
- The four populations are well separated in phase space leaving a gap between them.
- Current densities obey the Langmuir condition.
- The plasma is often subject to two-stream (i.e. Buneman) instabilities.

The first property is a direct consequence of the DL's strength  $\phi_{DL}$ . First consider particles of the decelerated populations. Any particle belonging to this population will have an entry velocity squared,  $v^2$ , larger than the square of a minimum penetration

velocity  $v_p^2 = 2e\phi_{DL}/m$  to be able to penetrate the DL. Dividing through with mass  $m/2$  on the strong DL assumption it is easy to see that

$$e\phi_{DL} \gg \kappa T \Rightarrow v_p^2 = \frac{2e\phi_{DL}}{m} \gg \frac{2\kappa T}{m} = 2v_{th}^2.$$

From the geometry of the DL, it follows that the probability density functions (PDFs) of reflected particles, must be symmetric around zero velocity. As particles forming the reflected and decelerated populations are assumed to have the same origin, they can be described by the same PDF at the boundary. The particles represented by this PDF are regarded as reflected or decelerated by the criteria given in section 3.2.

Assuming that the reflected/decelerated populations is described by a PDF representing a plasma in thermal equilibrium, more than 99% of the particles of specie  $s$  will have velocities  $v_s < 4v_{ths}$ . Given the strong double layer assumption, the velocities  $v \approx 4v_{ths}$  are lower than the required penetration velocity, hence no decelerated population is present. The lack of decelerated populations is illustrated in the phase space diagrams for strong DLs in figure 3.6. Each population of this figure can be seen as one of group dots, hence only two populations per specie are present, and only four all together.

The second property is a consequence of the Bohm criterion, derived in section 3.3.2, giving a lower limit for the entry velocity of the free particles. This lower limit sets a minimum distance between the populations in velocity space, seen as a gap in phase space diagrams of figure 3.6.

Assuming charge neutrality and therefore zero electric field at the boundaries, it can be proven that the current density ratio of the species equals  $j_i/j_e = \sqrt{m_e/m_i}$ , which is known as the Langmuir condition. The Langmuir condition was derived in section 3.3.3.

The fourth property comes from the fact that the gap mentioned as the second property is not a stable configuration for any plasma or fluid and will lead to two-stream instabilities, converting kinetic energy into electrostatic field energy. One of these instabilities, found to be highly relevant for DLs, is the Buneman instability, which was introduced in section 3.3.4.

### 3.4.2 Weak double layers

Weak DLs (*weak*) represent an asymptotic limit of DLs, different from the strong DLs (*strong*) in many ways. *Weak* are often a consequence of ion/electron acoustic waves, short lived and not as stable as the *strong*. Compared to the properties presented for *strong*, *weak* have decelerated populations present, the populations are not as separated in the phase space diagrams and not necessarily current dominated. The presence of the decelerated populations offer a possibility of the asymptotic limit of no net current through the DL. Assuming the densities and drift velocities of decelerated and accelerated populations are equal for the ions and electrons, the DL becomes *current free*.

Figure 3.6 introduces two types of *weak*: Slow Electron Acoustic DL (SEADL) and Slow Ion Acoustic DL (SIADL). Slow Electron (Ion) Acoustic DL's most striking feature is the "tuning fork" in the electron (ion) phase space diagram, thought to be a descendant of electron (ion) holes. SIADL is the only one of these two that can exist under current free conditions [Schamel, 1986].

Current-free DLs can form, e.g., at the boundary between two plasmas of different properties (temperature, density, etc.). Consider two plasmas of highly different temperatures separated by a fictional plane surface, where ions are assumed immobile in both plasmas. At the boundary, the flux of electrons from the hot plasma into the cold will be larger than in the opposite direction, hence a surplus of positive and negative charged particles will be formed on opposite sides of the boundary, in other words a DL. The

DL electric field will then balance the current produced by the temperature gradient by producing an oppositely directed current equal in strength. As long as the temperature gradient is present and the net current density remains zero, the double layer will remain stationary.

### 3.5 Theoretical models in one dimension

In the following, BGK-solutions [Bernstein et al., 1957], DL probability density functions, the water-bag model [Davidson, 1972] and the Sagdeev potential [Raadu, 1989; Smith, 1982] are introduced and used in a steady state description of a DL in one dimension [Jovanović et al., 1982]<sup>4</sup>.

#### 3.5.1 BGK solutions

Bernstein, Greene and Kruskal [Bernstein et al., 1957] showed that it is possible to solve the fully nonlinear steady state Vlasov's equations in at least one spatial dimension, solutions now referred to as "BGK-solutions". In this model, the Vlasov's equations (2.11) becomes

$$v \frac{\partial}{\partial x} f_s(x, v) - \frac{q_s}{m_s} \frac{\partial \phi(x)}{\partial x} \frac{\partial f_s(x, v)}{\partial v} = 0 ,$$

for each particle specie  $s$ . The assumption that a frame of reference exists, so that  $\phi(x)$  represents a stationary potential variation, is made. Given the density as the integral of  $f_s$  over  $v$ ;  $\int_{-\infty}^{\infty} f_s(x, v) dv = n_s(x)$ , Poisson's equation becomes

$$\frac{d^2 \phi(x)}{dx^2} = \frac{1}{\epsilon_0} \sum_s q_s \int_{-\infty}^{\infty} f_s \left( \frac{1}{2} m_s v^2 + q_s \phi(x) \right) dv.$$

As the model is one dimensional, effects from magnetic fields can be ignored, leaving the system as a purely electrostatic problem.

The study of [Bernstein et al., 1957] demonstrated that, assuming  $\phi(x)$  is given, it is possible to determine probability density functions for each particle population, so that when they are inserted into Poisson's equation they give precisely the given potential  $\phi(x)$ . The separation of particles into free and trapped populations is done as described in section 3.2. Note that the probability density functions of trapped particles must be symmetric in velocity space around  $v = 0$ !

For simplicity, assume that the only particle species present are electrons and singly charged positive ions, denoted by indices  $e$  and  $i$ . The particles energy  $\mathcal{E}_{e,i} = \frac{1}{2} m_{e,i} v^2 + q_{e,i} \phi(x)$  is introduced as a variable, further using that

$$dv = d\mathcal{E}_{e,i} / (m_{e,i} v) = d\mathcal{E}_{e,i} / \sqrt{2m_{e,i}(\mathcal{E}_{e,i} - q_{e,i} \phi(x))} ,$$

Poisson's equation can be rewritten by distinguishing free and trapped electron populations

$$\begin{aligned} \frac{d^2 \phi(x)}{dx^2} = & -\frac{e}{\epsilon_0} \int_{e\phi}^{\infty} \frac{f_i(\mathcal{E}_i)}{\sqrt{2m_i(\mathcal{E}_i - e\phi)}} d\mathcal{E}_i \\ & + \frac{e}{\epsilon_0} \int_{-e\phi_{min}}^{\infty} \frac{f_e(\mathcal{E}_e)}{\sqrt{2m_e(\mathcal{E}_e + e\phi)}} d\mathcal{E}_e \\ & + \frac{e}{\epsilon_0} \int_{-e\phi}^{-e\phi_{min}} \frac{f_e(\mathcal{E}_e)}{\sqrt{2m_e(\mathcal{E}_e + e\phi)}} d\mathcal{E}_e , \end{aligned} \quad (3.11)$$

<sup>4</sup>In strong magnetic fields the particles can be assumed to flow as "pearls on a string" and a one dimensional model may be applicable in such cases.

where  $\phi_{min}$  represent a local minimum of  $\phi(x)$ . Again for simplicity, assume that  $\phi_{min}$  represent the only local minimum of  $\phi(x)$ .

Probability density functions are prescribed for all populations except the trapped electrons, so that all terms of equation (3.11) are known except the last. Rewriting the equation as

$$\frac{e}{\epsilon_0} \int_{-e\phi}^{-e\phi_{min}} \frac{f_e(\mathcal{E}_e)}{\sqrt{2m_e(\mathcal{E}_e + e\phi)}} d\mathcal{E}_e \equiv G(e\phi) . \quad (3.12)$$

$G(e\phi)$  is thus a known function, containing all of equation (3.11), except the last integral for the trapped electrons. Note that equation (3.12) can be solved analytically. The so far unspecified probability density function can thus be found through equation (3.12), and together with the prescribed functions they form a set consistent with the given potential profile  $\phi(x)$ . An exact stationary solution for the fully non-linear coupled Vlasov-Poisson set of equations is thus found, with the explicit solution of equation (3.12) is

$$f_e(\mathcal{E}_e) = \frac{\sqrt{2m}}{\pi} \int_{e\phi_{min}}^{-\mathcal{E}_e} \frac{dG(U)}{dU} \frac{dU}{\sqrt{-\mathcal{E}_e - U}} \quad \text{for } \mathcal{E}_e < -e\phi_{min} , \quad (3.13)$$

where  $U \equiv e\phi$ .

This one dimensional model is not easily generalized to higher dimensions, so in most plasma phenomena, BGK-solutions are inadequate to describe the plasma. However, BGK-solutions have a good chance of being realized in strong magnetic fields where one dimensional models can be justified [Jovanović et al., 1982]. Although BGK-solutions in theory can describe an infinite number of solutions, only three have been observed in nature and lab experiments; double layers, ion- and electron-holes.

### 3.5.2 Probability density functions of a double layer

Any DL will by its own nature separate two different plasmas. The theory of BGK-solutions [Bernstein et al., 1957], with prescribed boundary probability density functions (PDFs) representing the two plasmas, are used to define a complete set of spatially varying PDFs. The full set of boundary PDFs for every particle specie (electrons/ions) and populations (accelerated/reflected/decelerated) are defined

- $f_{ea}(x_{lb}, v)$  - Accelerated electrons entering at the lower boundary
- $f_{er}(x_{ub}, v)$  - Reflected electrons entering at the upper boundary
- $f_{ed}(x_{ub}, v)$  - Decelerated electrons entering at the upper boundary
- $f_{ia}(x_{ub}, v)$  - Accelerated ions entering at the upper boundary
- $f_{ir}(x_{lb}, v)$  - Reflected ions entering at the lower boundary
- $f_{id}(x_{lb}, v)$  - Decelerated ions entering at the lower boundary

where  $x_{lb}$  and  $x_{ub}$  represent the lower and upper boundaries, respectively.

As with BGK-solutions in general, the potential profile  $\phi(x)$  is assumed to be given, with the low potential at the lower boundary and high potential at the upper boundary. The energy of any particle can then be given by the equation

$$\mathcal{E}_s = \frac{1}{2} m_s v^2(x_i) + q_s \phi(x_i) = \frac{1}{2} m_s v^2(x) + q_s \phi(x) .$$

The above result gives the trajectory  $v(x)$  of a particle in phase space, with initial velocity  $v(x_i)$  under the influence of the spatial varying potential  $\phi(x)$ . With the result for  $v(x)$  as given above, the PDFs of both space and velocity becomes

$$f_{sp}(x, v) \equiv f_{sp}(x_i, v(x_i)) = f_{sp} \left( x_i, \pm \sqrt{v^2(x) + \frac{2q_s}{m_s}(\phi(x) - \phi(x_i))} \right), \quad (3.14)$$

for particle population  $p$  and specie  $s$  entering the DL at  $x = x_i$ . Integrating the PDFs over velocity space yields the particle density

$$n_{sp}(x) = \int f_{sp}(x, v) dv, \quad (3.15)$$

of each specie and population. As for general BGK-solutions [Bernstein et al., 1957], the charge density retrieved from the density profiles of equation (3.15), when inserted into Poisson's equation yield the given potential profile  $\phi(x)$ . This procedure is repeated in section 3.5.5, where the potential profile is given by a hyperbolic tangent function and the PDFs are assumed Maxwellian.

### 3.5.3 Water-bag models

An interesting choice of PDFs are functions which are either stepwise constant or zero, a model often referred to as a *water-bag* model [Davidson, 1972]. Compared to a Maxwellian distribution, the water-bag equivalent would be a rectangular shaped box of width  $2\sigma$  ( $\sigma$  is the variance of the Maxwellian) and functional value  $f_0$  so that the integral of each PDF over the entire velocity space equals the density.

A phase space description of a plasma now simplifies to the contours separating the non-zero from the zero regions. From water-bag theory [Davidson, 1972], these contours are found through the differential equation

$$\frac{\partial}{\partial t} U_j + U_j \frac{\partial}{\partial x} U_j = \frac{e}{m} \frac{\partial}{\partial x} \phi$$

which when assuming stationarity simplifies to

$$\frac{\partial}{\partial x} U_j^2 = \frac{2e}{m} \frac{\partial}{\partial x} \phi \quad \text{for } j = 1, 2, 3, \dots$$

The above differential equations have the solutions

$$U_j = \pm \sqrt{w_j + 2 \frac{e}{m} \phi} \quad \text{for } j = 1, 2, 3, 4 \quad (3.16)$$

for a given  $\phi = \phi(x)$  typical for BGK-solutions. Due to the opposite charge of electrons and ions, the two particle species experience forces in the opposite directions from  $\phi(x)$ , hence a reversal of the form  $\psi = \psi(x) = \phi_{DL} - \phi(x)$  will account for the opposite charge. Given that the contours in equation (3.16) apply for electrons,

$$U_{ej} = \pm \sqrt{w_{ej} + 2 \frac{e}{m} \phi} \quad \text{for } j = 1, 2, 3, 4$$

the ion contours becomes

$$U_{ij} = \pm \sqrt{w_{ij} + 2 \frac{e}{m} \psi} \quad \text{for } j = 1, 2, 3, 4.$$

Assuming that  $\phi(x)$  represents a DL, appropriate choices for the contours are

$$\begin{array}{cccc} w_{e1} > 0 & w_{e2} = 0 & w_{e3} = 0 & w_{e4} > 0 \\ w_{i1} > 0 & w_{i2} = 0 & w_{i3} = 0 & w_{i4} \geq 0 \\ U_{e1} \leq 0 & U_{e2} \leq 0 & U_{e3} \geq 0 & U_{e4} > 0 \\ U_{i1} < 0 & U_{i2} \leq 0 & U_{i3} \geq 0 & U_{i4} \geq 0 \end{array} \quad (3.17)$$

allowing the following water-bag DL PDFs for electrons

$$f_{ea}(x, v) = \begin{cases} f_0 & \text{for } U_{e3} < v \leq U_{e4} \\ 0 & \text{otherwise} \end{cases}, \quad (3.18)$$

$$f_{er}(x, v) = \begin{cases} f_0 & \text{for } U_{e2} \leq v \leq U_{e3} \\ 0 & \text{otherwise} \end{cases}, \quad (3.19)$$

$$f_{ed}(x, v) = \begin{cases} f_0 & \text{for } U_{e1} \leq v < U_{e2} \\ 0 & \text{otherwise} \end{cases}, \quad (3.20)$$

and for ions

$$f_{ia}(x, v) = \begin{cases} f_0 & \text{if } U_{i1} \leq v < U_{i2} \\ 0 & \text{otherwise} \end{cases}, \quad (3.21)$$

$$f_{ir}(x, v) = \begin{cases} f_0 & \text{if } U_{i2} \leq v \leq U_{i3} \\ 0 & \text{otherwise} \end{cases}, \quad (3.22)$$

$$f_{id}(x, v) = \begin{cases} f_0 & \text{if } U_{i3} < v \leq U_{i4} \\ 0 & \text{otherwise} \end{cases}. \quad (3.23)$$

The contours for  $j = 2, 3$  can also be recognized as the separatrix velocities setting the boundary between *free* and *trapped* particles, defined as  $V_e(x) = \sqrt{2e\phi(x)/m_e}$  and  $V_i(x) = \sqrt{2e\psi(x)/m_i}$ . Although this is a fairly limited picture of the probability distribution functions it works as a highly efficient tool for simple analysis.

Due to the form of the velocity distribution function, it is now a simple matter to obtain the density (equation (3.15)). For instance, for the accelerated electron population the density becomes

$$n_{ea}(x) = (U_{e1} - U_{e2})f_0,$$

where  $f_0$  is as defined above.

### 3.5.4 The Sagdeev potential

The Sagdeev potential<sup>5</sup> [Smith, 1982; Raadu, 1989]

$$V(\phi) \equiv - \int_0^\phi \rho(\phi) d\phi, \quad (3.24)$$

gives a highly useful tool in the theoretical work in plasma physics and an analytical solution to Poisson's equation (2.6) in one spatial dimension. It also serves as a tool for checking if the total charge density is zero, meaning that  $V(\phi_{DL}) = 0$  must be upheld. By the assumption applied in (3.24), the Sagdeev potential presumes that the charge density somehow is expressed as a function of the potential  $\phi$ . This will be illustrated by examples later on.

---

<sup>5</sup>Also known as the classical or pseudo potential

Poisson's equation with the Sagdeev potential introduced can be written with solution

$$\frac{d^2\phi}{dx^2} = -\frac{\rho}{\epsilon_0} \equiv \frac{dV(\phi)}{d\phi} \quad (3.25)$$

$\Downarrow$

$$\left(\frac{d\phi}{dx}\right)^2 - V(\phi) = C, \quad (3.26)$$

where  $C$  is an integration constant, often choosing  $C = 0$  which limits the physical range of  $V$  to  $V \geq 0$ . Integrating equation (3.26) [Raadu, 1989] gives the relation

$$x(\phi) - x_0 = \pm \sqrt{\frac{1}{2\alpha}} \int_{\phi_0}^{\phi} \frac{d\phi}{\sqrt{C + V(\phi)}}. \quad (3.27)$$

By introducing an adjustment of the charge density amplitude by a factor  $\alpha$ ,  $\rho'(\phi) = \alpha\rho(\phi)$  yields an adjusted Sagdeev potential  $V'(\phi) = \alpha V(\phi)$ . Further assuming  $\phi(x) \in [0, \phi_{DL}]$ ,  $x \in [0, \ell]$ , where  $\ell$  and  $\phi_{DL}$  known, the constant  $\alpha$  can be calculated through the equation

$$\ell = \frac{1}{\sqrt{2\alpha}} \int_0^{\phi_{DL}} \frac{d\phi'}{\sqrt{V(\phi')}} , \quad (3.28)$$

and so adjusting the level of  $\rho$  to coincide with any chosen set of DL strength  $\phi_{DL}$  and width  $\ell$ .

### 3.5.5 Examples in one dimension

In previous sections, the theoretical foundation for describing and analysing a DL has been introduced. As an example, mathematical expressions are inserted for the potential profile and boundary PDFs;  $\phi(x) \sim \tanh(x)$  and  $f(x, v) \sim \exp(-\beta v^2)$ . The latter can be recognized as a Maxwellian distribution function with the appropriate choice of  $\beta$ . The exact form of the potential profile is

$$\phi(x) = \frac{\tanh(x - \ell/2) + 1}{2} \phi_{DL}, \quad \text{for } x \in [x_{lb}, x_{ub}], \quad (3.29)$$

with  $x_{lb} < 0 < \ell < x_{ub}$ . The corresponding electric field and charge density are found through Poisson's equation (2.6) by taking the first and second derivative of equation (3.29) yields

$$E(x) = -\frac{d\phi(x)}{dx} = -\frac{1}{\cosh^2(x - \ell/2)} \frac{\phi_{DL}}{2},$$

$$\rho(x) = -\frac{d^2\phi(x)}{dx^2} = \frac{\tanh(x - \ell/2)}{\cosh^2(x - \ell/2)} \phi_{DL}.$$

Plots of these are shown in figure 3.7 together with  $\phi(x)$ .

Probability density functions at the boundaries have, as mentioned earlier, the mathematical shape of Maxwellian distribution functions. In the present discussions, the DL is assumed to strong, therefore no decelerated populations are present. The exact form of the exponentials are given in equations (3.30)-(3.33). Here indices  $e$  and  $i$  refer to electron and ion species,  $a$  and  $r$  refer to accelerated and reflected populations, and  $u_e > 0$  and  $u_i < 0$  are the drift velocities of accelerated electrons and ions, respectively.

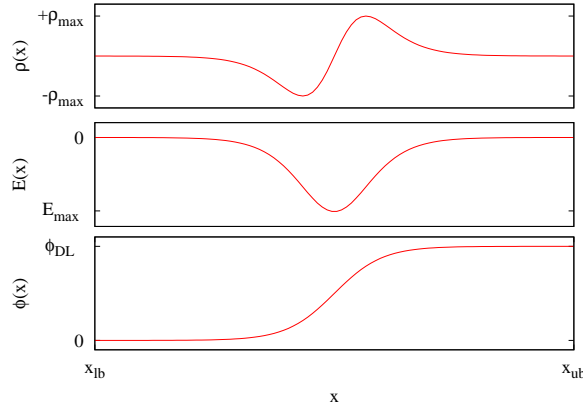


Figure 3.7: Charge density, potential and electric field in configuration space when assuming a hyperbolic tangent potential structure (3.29).

$$f_{ea}(x_{lb}, v) \sim \exp\left(-\frac{(v - u_e)^2}{2v_{the}^2}\right) \quad (3.30)$$

$$f_{er}(x_{ub}, v) \sim \exp\left(-\frac{v^2}{2v_{the}^2}\right) \quad (3.31)$$

$$f_{ia}(x_{ub}, v) \sim \exp\left(-\frac{(v - u_i)^2}{2v_{thi}^2}\right) \quad (3.32)$$

$$f_{ir}(x_{lb}, v) \sim \exp\left(-\frac{v^2}{2v_{thi}^2}\right) \quad (3.33)$$

Applying the transformation of the boundary PDFs, given by the theory of BGK-solutions, to the two phase space dimensions (equation (3.14)), gives the full PDF description of a steady state DL. A phase space representation of these are plotted in figure 3.8.

As argued in section 3.5.1, the density retrieved from any set of PDFs meant to represent a DL must be the source of the same charge density profile responsible for the assumed potential profile (equation (3.29)). Beginning with the assumed potential profile, leading to the PDFs given in equations (3.30)-(3.33) the particle density of each population and specie  $n_{sp}(x)$  is retrieved through integration of the PDFs over velocity space.

The reflected population PDFs are easily integrated (centered Maxwellian) and yield Boltzmann distributions

$$n_{sr}(x) \sim \exp\left(-\frac{e(\phi(x) - \phi(x_i))}{\kappa T_s}\right)$$

for both reflected populations. Equations (3.30) and (3.32) are not as easily integrated analytically (must be done numerically), hence an exact analytic expression for the density is not retrieved here. However, from a conservation of flux perspective it is possible to determine the trend of the density profile.

Conservation of flux ( $\int v f(x, v) dv = \text{constant}$ ) states that density is proportional to the inverse of the bulk velocity of the plasma (subdivided into population and specie),



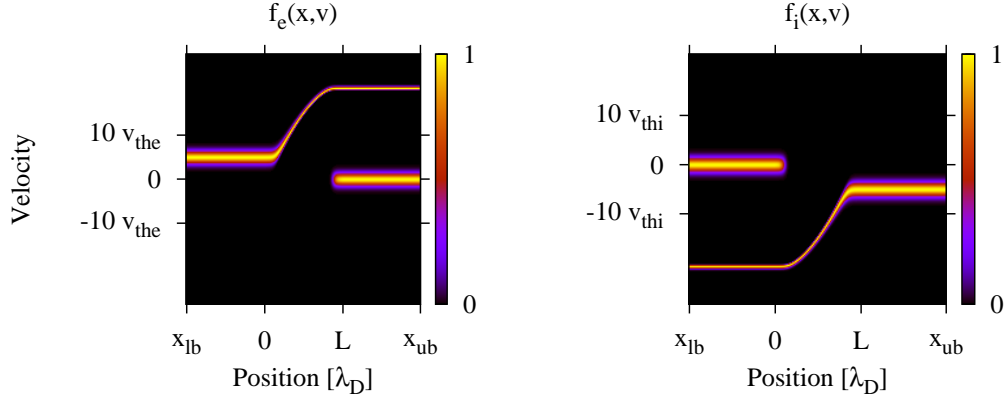


Figure 3.8: *Phase space representation of PDFs for electrons (left; equation (3.30)-(3.31)) and ions (right; equation (3.32)-(3.33))*

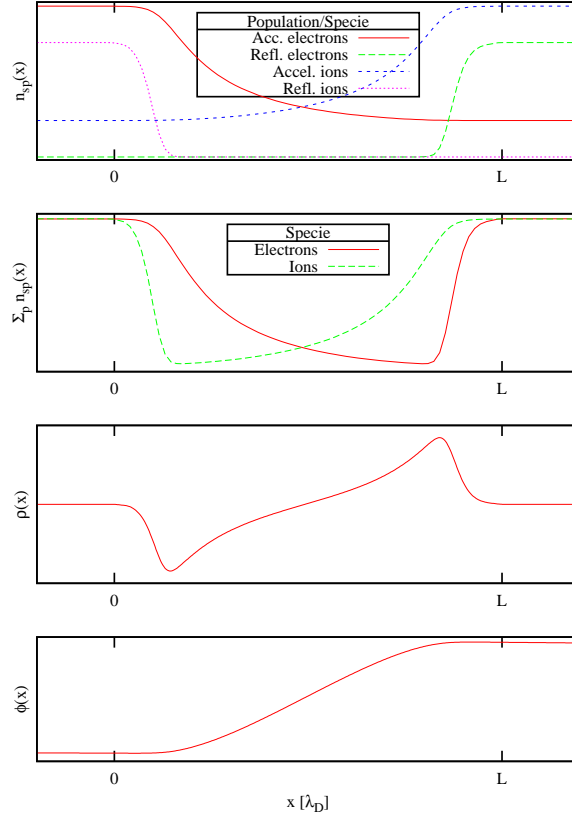


Figure 3.9: *Typical density profiles of a DL (first and second panel), with corresponding charge density (third panel) and potential profile (fourth panel).*

hence

$$n_{sa}(x) \sim \frac{1}{u_{sa}(x)} = \left( u_{sa}^2(x_i) + \frac{2q_s}{m}(\phi(x) - \phi(x_i)) \right)^{-1/2}.$$

Taking the ratio between the two latter equations it is easy to show that the reflected populations decrease more rapidly in density than the accelerated populations at both boundaries. This effect causes charge separation necessary to create the DL charge density. Figure 3.9 shows such an example, consistent with the assumed  $\phi(x)$  (3.29).

Assuming, on the other hand, the Water-bag model PDFs, as they are introduced in section 3.5.3. With the given potential profile, equation (3.29), the four Water-bag contours for electrons are as illustrated in the phase space representation in figure 3.10. Here, suitable choices for the constants  $w_{sj}$  are made, corresponding to a typical DL set of PDFs, with the decelerated present. The areas between the contours have thus constant functional value of  $f_0$ . Similar contours for ions are retrieved by mirroring this figure around both axes.

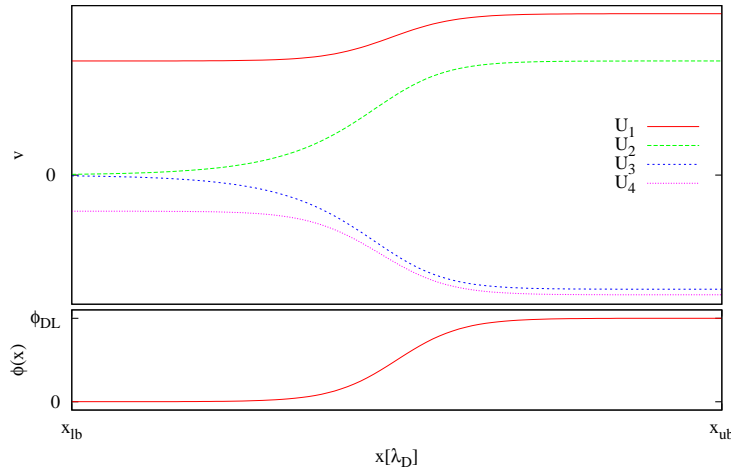


Figure 3.10: *DL water-bag contours satisfying the criteria set in equation (3.17). In this example, which applies for electrons  $w_{e1} = 2$  and  $w_{e4} = 0.2$ . The lower panel show the corresponding  $\phi(x)$ .*

The Sagdeev potential, introduced in section 3.5.4 is found from the charge density given as a function of  $\phi$ . Starting with the charge density as a function of configuration space, retrieved from the assumed potential profile (3.29). Then inverting  $\phi(x)$ , and introducing  $x_m = x - \ell/2$ , yields

$$\begin{aligned} \rho(x_m) &= \frac{\tanh(x_m)}{\cosh^2(x_m)} \phi_{DL} = \tanh(x_m)(1 - \tanh^2(x_m)) \phi_{DL}, \\ \tanh(x_m) &= \frac{2\phi}{\phi_{DL}} - 1. \end{aligned}$$

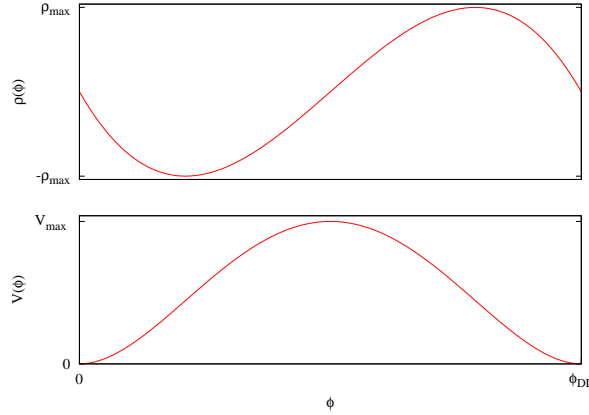


Figure 3.11: Charge density as a function of potential  $\rho(\phi)$  and the corresponding Sagdeev potential  $V(\phi)$  when assuming the hyperbolic tangent potential profile (3.29).

Inserting the latter into the prior yields

$$\begin{aligned}\rho(\phi) &= \left( \frac{2\phi(x)}{\phi_{DL}} - 1 \right) \left( 1 - \left( \frac{2\phi(x)}{\phi_{DL}} - 1 \right)^2 \right) \phi_{DL} \\ &= 4\phi \left( \frac{2\phi}{\phi_{DL}} - 1 \right) \left( 1 - \frac{\phi}{\phi_{DL}} \right).\end{aligned}$$

The expression for  $\rho(\phi)$  is a polynomial, thus it is easily integrated and the Sagdeev potential can be obtained through equation (3.24)

$$V(\phi) = - \int_0^\phi \rho(\phi) d\phi = \phi_{DL} (2\xi^4 - 4\xi^3 + 2\xi^2)$$

where  $\xi \equiv \phi/\phi_{DL}$ . Both the charge density and the Sagdeev potential in this example are plotted in figure 3.11.

### 3.6 Numerical studies of double layers

The development of the code for the present numerical simulations is based on previous numerical studies summarized in the review article [Smith, 1982]. It is therefore natural to present here the characteristics of these previous studies, both as a preliminary discussion and to form a basis of comparison for the results of the present simulations.

Table 3.1 presents six of the cited previous studies found in [Smith, 1982] whereas the last line of this table contains details of the present simulations. The latter will be discussed later in this section and in chapter 5. All studies listed in table 3.1 simulate DLs in the Buneman regime (i.e. strong DLs) in one spatial dimension. The characteristics of the listed studies are presented in columns 2-8, the contents of which can be interpreted as in the following text.

The simulation schemes, specified in the second column, are in this case either Particle-in-Cell (PIC) or integration of Vlasov's equations (Vlasov). The details of these schemes were presented in section 2.4.

The boundary conditions (BCs) on the potential is specified in the third column of table 3.1 where the notation is of the form "left BC"- "right BC". D or vN refer to

Col. no.:	2	3	4	5	6	7	8
Ref.	Scheme	Pot. BC	Init	$L_{sys}$	$\phi_{DL}$	$U$	$N_s$
HuJo	PIC	D,vN- $\emptyset$	DL	[30,360]	[10,200]	$\sim 2.5$	$10^4$
Si	Vlasov	D-D	Void	[50,100]	[20,30]	[0.4,1.5]	N/A
Jo	Vlasov	D-D	Void	50	[16,30]	[9,16]	N/A
GoJo	PIC	D,vN- $\emptyset$	Hom	[37,100]	N/A	[0.5,1.5]	$10^4$
Sm	Vlasov	vN-vN	DL	[20,57]	20	2	N/A
Be	PIC	<i>floating</i>	Hom	100	270	1.8	N/A
Pr	PIC	$\dagger$	DL	[550,1250]	[25,800]	[1.8,9]	$10^7$

(a)

Abbreviation	Reference
HuJo	[Joyce and Hubbard, 1978; Hubbard and Joyce, 1979]
Si	[Singh, 1980]
Jo	[Johnson, 1980]
GoJo	[Goertz and Joyce, 1975]
Sm	[Smith, 1982]
Be	[Belova et al., 1980]
Pr	Present studies

(b)

Table 3.1: *a) Previous studies on one dimensional double layers in the Buneman regime as listed in table 1 of [Smith, 1982] (N/A: None Available). The topmost line is merely a numbering of columns. b) Abbreviations to article citations used in a).*

Dirichlet or von Neumann type boundaries, respectively,  $\emptyset$  means none is given. The  $\dagger$ -symbol in the BC column of present studies corresponds to "D-D, D-vN, vN-D and vN-vN". In the case of [Belova et al., 1980], the exact type of boundary conditions was not specified, except that they were *floating*, meaning the BCs include at least one boundary without a Dirichlet type condition.

The initialized plasma state, shown in table 3.1, fourth column, are in this case either Homogeneous plasma (Hom), empty simulation domain (Void) or a DL. Studies based on Void and Hom type initialization focus on whether a DL is able to build in the initialized plasma, while simulations with DL initialization type study the stability of the initialized DL.

The last four columns of table 3.1 contain typical parameters used to describe the characteristics of the double layers, set to a single value or range of values as specified. The parameters are (with units given in parentheses) system length,  $L_{sys}$  ( $\lambda_{De}$ ), potential leap/strength,  $\phi_{DL}$  ( $\kappa T/e$ ), drift velocity of accelerated particles of specie  $s$ ,  $U$  ( $v_{ths}$ ), and number of particles per specie  $N_s$ .

### Previous studies

All the system lengths,  $L_{sys}$ , of the listed previous studies lie within the range [20, 360]. Relative to the system lengths, the width of the simulated DLs fill  $\sim 30\%$  for [Joyce and Hubbard, 1978; Hubbard and Joyce, 1979] and [Johnson, 1980],  $\sim 60\%$  for the remaining. In all the previous studies, the DL widths are natural saturations of the respective simulations and not prescribed parameters by the respective programs.

The finite potential difference (leap) of the DL,  $\phi_{DL}$ , is fixed (D-D) for [Singh, 1980] and [Johnson, 1980], floating otherwise, as is specified in the column for boundary con-

ditions (third column). For the studies with floating potential leap, the specified ranges are thus natural saturations of the respective simulations.

The choices of drift velocities  $U$  of the accelerated particle populations divide the previous studies into two groups; those with all velocities higher than the critical velocity of the Buneman instability, and those with velocities both lower and higher than this critical value. In the latter group [Singh, 1980] and [Goertz and Joyce, 1975] can be found. Both these studies confirm the effect presence of a critical value for the Buneman instability, as DLs were not able to form unless  $U \geq 1.5$ .

Where the number of simulation particles,  $\mathcal{N}_s$ , were specified, they were in order of magnitude equal to  $10^4$ . Most of the previous studies did not provide this information, however, for those simulating with a Vlasov scheme, there is no such thing as particles, hence none was given.

### Present studies

The boundary conditions used in the present studies are the combinations D-D, D-vN, vN-D and vN-vN. The values given to either the potential (Dirichlet) or derivative of the potential (von Neumann) at the left and right hand side of the DL are as given in table 3.2.

	left	right
Dirichlet	$\phi = 0$	$\phi = \phi_{DL}$
von Neumann	$d\phi/dx = 0$	$d\phi/dx = 0$

Table 3.2: *The values set to either potential or the derivative of the potential, in the case of Dirichlet or von Neumann type boundary conditions, at left and right hand side of the DL.*

Present studies have performed simulations with system lengths by a factor of 10 larger than what was done in the previous studies. The previous studies suffered from lower memory in their computers, than what is available today. Therefore they did not include the plasmas outside the DL in the simulations, as this meant to increase the particle numbers, and so increasing memory usage remarkably. The studies of [Joyce and Hubbard, 1978; Hubbard and Joyce, 1979] and [Johnson, 1980] are exceptions to this rule, as the DLs they studied filled only 30% of the system length. Present studies were able to improve on this detail, and have added homogeneous plasmas to each side of the DL, each of width  $250\lambda_{De}$ , in all of the present simulations. Hence, the system length equals  $L_{sys} = L + 500\lambda_{De}$  where  $L$  is defined as the DL width. Hence, the variation of DL width is  $L \in [50, 750]$  which for the lower part of this range, is within the specified ranges for previous studies.

In present studies, the specified range of the potential leap was prescribed for each simulation through the initialization of the DL. The range was initially chosen equal to that of the previous, then later increased to investigate even stronger DLs. Drift velocities were chosen higher than what is predicted to trigger the Buneman instability [Singh, 1980; Goertz and Joyce, 1975], hence a Buneman regime DL was studied.

All previous studies operate with fictional ion masses in the ranges  $m_i/m_e \in [16, 64]$  with the exception of [Joyce and Hubbard, 1978; Hubbard and Joyce, 1979] who used  $m_i/m_e = 256$  in some of their simulations. Present simulations use mass ratios of  $m_i/m_e \in [1, 300]$ . The temperature ratio  $T_i/T_e$  is set to unity in all previous studies with the exception of [Singh, 1980] where  $T_i/T_e = 0.5$ . In the present study, the temperature ratio is set to unity.

In addition to the simulation details given in table 3.1, previous and present studies can be summarized as follows:

- 
- I The spatial and temporal resolution of present simulations:
    - (a) The spatial step length is set  $dx \approx 0.5\lambda_{De}$  to satisfy the stability requirements of the Particle-in-Cell method [Birdsall and Langdon, 1985].
    - (b) The temporal step length is set  $dt \approx 0.6(dx/v)$ . Here  $v$  is the largest possible velocity of any particle in the initialized state. The condition is often referred to as the *Courant, Friedrich and Lewy condition* [Courant et al., 1928].
  - II Important similarities between previous and present studies:
    - (a) Boundaries are open, thus allowing particles to enter and leave the simulation domain.
    - (b) The assumption of prescribed distributions at the boundaries.
    - (c) Boundary probability density functions are shifted and centered Maxwellians for accelerated and reflected populations, respectively.
    - (d) Current densities obey the Langmuir condition.
  - III Important changes in the present studies as compared to the previous:
    - (a) Larger DL widths.
    - (b) Higher particle numbers by a factor of  $10^3$ .
    - (c) Several possible combinations for boundary conditions opening for more detailed study of how boundary conditions influence the result.

## Chapter 4

# Numerical model

Section 2.4 introduced numerical simulations as an important tool in plasma physics. Combining this with the plasma theory presented in chapters 2 and 3, a complete numerical model of the plasma can be developed.

The present simulations are performed with a Particle-in-Cell code in one spatial dimension, that was developed as a part of this thesis. Transformations of random numbers into the desired probability density functions is performed with the Inversion method [Trulsen, 2005]. The previous numerical studies discussed in section 3.6, form a basis for which the present numerical model was meant to be comparable with.

In this chapter, the equations of motion, field equations and flux equations, with the quantities they contain, are made dimensionless and discretized. The Inversion method is introduced, together with the algorithms of the present numerical model. The complete model is tested for two plasma conditions. One plasma homogeneous in space and in thermal equilibrium, and one containing a double layer. Finally, the diagnostic routines developed for the simulations are presented in the last sections.

### 4.1 General assumptions and approximations

In present simulations, the following approximations and assumptions have been made:

- The plasma is assumed to be one dimensional. To some extent this is true for magnetic flux tubes [Jovanović et al., 1982], typical for magnetospheric physics. In this model the magnetic field is either assumed non existing ( $\mathbf{B} = \mathbf{0}$ ) or parallel to the simulation axis  $x$ .
- The fields are electrostatic.
- There are no collisions
- Ionization and recombination processes do not occur.

### 4.2 Dimensionless equations and quantities

Any programming language offers only a limited level of precision when representing numbers. The highest such precision is often referred to as *double precision*. When dealing with numbers close to the given the precision, relative error of the simulation and corresponding results will increase. To avoid this, the quantities of the equations are normalized by constants (with the same units as the quantities), typical for the physical

phenomena simulated. Through the normalization, the equations and their quantities are made dimensionless and the involved numbers are less likely to approach the allowed precision.

With the assumptions given in 4.1, the equations in chapters 2 and 3 used in the numerical model are made dimensionless through suitable choices of normalization factors. Throughout this chapter, the notation

$$f' = \frac{f}{\mathcal{F}} ,$$

is used, where  $f$  is the physical quantity with dimensions,  $f'$  it's dimensionless equal, scaled by the normalization factor  $\mathcal{F}$ . Note that  $f'$  does *not* mean the derivative of a function  $f$  as is usual notation in other literatures.

### 4.2.1 Equations of motion

The electron Debye length  $\lambda_{De}$ , electron thermal velocity  $v_{the}$  and the inverse of the electron plasma frequency  $\omega_{pe}^{-1}$  are natural choices of scaling factors for position, velocity and time, leading to the dimensionless variables

$$x' = \frac{x}{\lambda_{De}} , \quad v' = \frac{v}{v_{the}} \quad \text{and} \quad t' = t\omega_{pe} ,$$

while charge and mass terms are scaled with the elementary charge and electron mass, giving

$$q' = \frac{q}{e} \quad \text{and} \quad m' = \frac{m}{m_e} .$$

Inserting these into the equations of motion, equations (2.7) and (2.8), together with the assumptions of section 4.1 yields

$$\frac{dx'}{dt'} = \frac{v_{the}}{\lambda_{De}\omega_{pe}} v' = v' , \tag{4.1}$$

$$\frac{dv'}{dt'} = \frac{1}{v_{the}\omega_{pe}} \frac{q}{m} E = \frac{(e/m_e)E_0}{v_{the}\omega_{pe}} \frac{q'}{m'} E' = \frac{q'}{m'} E' . \tag{4.2}$$

The quantity  $E_0$  is defined together with the proof of  $\frac{(e/m_e)E_0}{v_{the}\omega_{pe}} = 1$  in the next section.

### 4.2.2 Field equations

Introducing  $\phi_0 = \kappa T_e/e$ ,  $E_0 = \phi_0/\lambda_{De}$  and  $\rho_0 = en_0$  as scaling factors for the electrostatic potential, electric field and charge density, where  $n_0$  is as a constant background reference density<sup>1</sup> yields the dimensionless quantities

$$\phi' = \frac{\phi}{\phi_0} , \quad E' = \frac{E}{E_0} \quad \text{and} \quad \rho' = \frac{\rho}{\rho_0} .$$

Inserting these into Poisson's equation in the general form (2.2) and in the electrostatic approximation (2.6), with the assumptions from section 4.1, yields the dimensionless field equations

$$\frac{d^2\phi'}{dx'^2} = -\rho'_e , \tag{4.3}$$

$$\frac{d\phi'}{dx'} = -E' . \tag{4.4}$$

---

<sup>1</sup>In the case of immobile ions,  $n_0$  equals the ion density.



The charge density is found through the Particle-in-Cell method discussed in section 2.4.1 and 4.5.1. In previous section, the equality  $\frac{(e/m_e)E_0}{v_{the}\omega_{pe}} = 1$  was left unproven; as  $E_0 = \phi_0/\lambda_{De}$ , it is easy to show that

$$\frac{(e/m_e)E_0}{v_{the}\omega_{pe}} = \frac{(e/m_e)(\kappa T/e)}{v_{the}\omega_{pe}\lambda_{De}} = \frac{v_{the}}{\lambda_{De}\omega_{pe}} = 1 .$$

### 4.2.3 Flux density

The directional flux of particles through a surface<sup>2</sup>, where the velocity distribution function  $f(v)$  is known, is given by the integral

$$P = \pm \int_0^{\pm\infty} v f(v) dv \quad (4.5)$$

choosing the negative or positive signs (in front of the integral and for the upper limit) depending on the direction of the flow ( $v \lesseqgtr 0$ ).<sup>3</sup>  $P$  can also be interpreted as the temporal change in the number of particles  $\mathcal{N}$  in either of the volumes separated by the surface,  $P = d\mathcal{N}/dt$ . Given a one dimensional model, the dimensions of  $P$  is expected to be  $s^{-1}$ . Dimensional studies then show that the dimensions of  $f(v)$  must be  $(m_s^{\frac{m}{s}})^{-1}$  such that  $f(v)dx dv$  is dimensionless. Space is excluded as a parameter of  $f$  as  $f$  is assumed homogeneous in space at the surface and therefore denoted only with  $v$  as a parameter.

Introducing the dimensionless quantities

$$dt' = \omega_{pe} dt, \quad v' = \frac{v}{v_{the}}, \quad dv' = \frac{dv}{v_{the}} \text{ and } f'(v') = f(v) \lambda_{De} v_{the}$$

yields the dimensionless expression for flux

$$\begin{aligned} P' = \frac{dN}{dt'} &= \pm \frac{v_{the}^2}{\omega_{pe} \lambda_{De} v_{the}} \int_0^{\pm\infty} v' f'(v') dv' \\ &= \pm \int_0^{\pm\infty} v' f'(v') dv'. \end{aligned} \quad (4.6)$$

The particles entering through a surface have velocities distributed according to the function  $vf(v)$ . As  $f$  is assumed homogeneous in space at the surface, the position of a flux particle is given by  $x = vdt \cdot R$  where  $R$  is a random number drawn from the uniform distribution  $U[0, 1]$  and  $v$  is drawn from the distribution  $vf(v)$ .

## 4.3 Discretized equations

In this section, the dimensionless equations of the previous section are discretized. Note that the prime (') is omitted from any step size. For instance,  $\Delta t$  is dimensionless and discretized equivalent to  $dt'$ .

### 4.3.1 Equations of motion

Orbit integration, i.e. integrating the equations of motion (4.1) and (4.2), is performed with the Leapfrog algorithm

$$a'_j = a'(x'_j) \quad (4.7)$$

$$v'_{j+1/2} = v'_{j-1/2} + a'_j \Delta t \quad (4.8)$$

$$x'_{j+1} = x'_j + v'_{j+1/2} \Delta t \quad (4.9)$$

<sup>2</sup>When the simulation is in one spatial dimension, such a surface is reduced to a point.

<sup>3</sup>Implicit in equation (4.5) is the assumption that particles arrive at the boundary from one direction only.

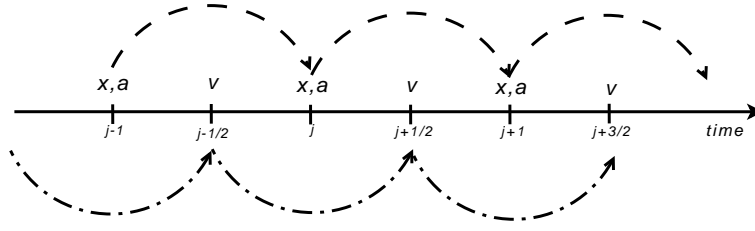


Figure 4.1: *Illustration of the leapfrog method with its staggered time steps in integration of the velocity. Time relates to the index  $j$  as  $t = j\Delta t$ .*

where  $t = j\Delta t$ . In Leapfrog, the velocities are evaluated in staggered time steps ( $t - \Delta t/2$ ,  $t + \Delta t/2$ ,  $t + 3\Delta t/2$ , ...) while positions and accelerations are evaluated in the normal time steps ( $t$ ,  $t + \Delta t$ ,  $t + 2\Delta t$ , ...), as illustrated in figure 4.1.

The Leapfrog algorithm is accurate to  $\mathcal{O}(\Delta t^2)$ , a result found through Taylor expansion of equations (4.8) and (4.9). Performing Taylor expansions on alternative orbit integration schemes like Euler's method, 2nd and 4th order Runge Kutta method leads to the accuracies  $\mathcal{O}(\Delta t)$ ,  $\mathcal{O}(\Delta t^2)$  and  $\mathcal{O}(\Delta t^4)$  respectively. Note that these methods perform the number of force calculations per time step, as specified in parentheses: Euler's (1), Leapfrog (1), Runge Kutta 2nd order (2) and Runge Kutta 4th order (4). Taking this into consideration, the following conclusions can be made:

Euler's and Leapfrog method are the fastest, while Leapfrog have better accuracy compared to Euler's. The 2nd order Runge Kutta method equals, while the 4th order Runge Kutta exceeds the accuracy of the Leapfrog algorithm. However, both Runge Kutta methods perform more force calculations than Leapfrog and are therefore slower. Hence, from a direct comparison of these four methods it is clear that Leapfrog has many advantages over the three others, given an accuracy of  $\mathcal{O}(\Delta t^2)$  is sufficient. However, there are two disadvantages with the Leapfrog algorithm. The first is the need for initialization, performed through back stepping the velocities by a half time step, the other the need for synchronization of velocities with position and acceleration in diagnostic routines.

### 4.3.2 Field equations

Inserting the standard discretization of single and double derivatives

$$\begin{aligned} \frac{df}{dx} &\approx \left. \frac{df}{dx} \right|_{i+1/2} = \frac{f_{i+1} - f_i}{\Delta x}, \\ \frac{d^2f}{dx^2} &\approx \left. \frac{d^2f}{dx^2} \right|_i = \frac{f_{i+1} + f_{i-1} - 2f_i}{(\Delta x)^2}. \end{aligned}$$

into equations (4.3) and (4.4), yields

$$\phi'_{i+1} + \phi'_{i-1} - 2\phi'_i = -\rho'(\Delta x)^2, \quad (4.10)$$

$$E'_{i+1/2} = \frac{\phi'_{i+1} - \phi'_i}{\Delta x}, \quad (4.11)$$

which can be recognized as the as the discretized Poisson's equations in it's general form and in the electrostatic field assumption.

## 4.4 Statistical methods

Probability density functions and statistical methods are both important factors in plasma physics. Present simulations use *the Inversion method* to generate particles according to the specified probability density functions. Particles are generated during the initialization of the plasma, prior to temporal simulations and during temporal simulations for the influx particles.

This section presents the Inversion method, derived from the Fundamental theorem [Trulsen, 2005] and a special case of the Inversion method called the Box-Muller algorithm. The Inversion method can transform between probability density functions of arbitrary type, while the Box-Muller method transform a uniform distribution into a normal distribution.

### 4.4.1 The Inversion method

The Fundamental theorem [Trulsen, 2005] introduces methods of transforming the random variable  $x$  drawn from a probability density function  $f_x$ , to a new random variable  $y = g(x)$  drawn from a probability density function  $f_y$ . The Inversion method can be derived from the Fundamental theorem, through restricting  $g(x)$  to be either strictly increasing or decreasing so that the transformation from  $x$  to  $y = g(x)$  only has one solution.

Further introducing the cumulative distributions

$$\mathcal{P}(x \leq x) \equiv F_x(x) = \int_a^x f_x(x) dx ,$$

where  $\mathcal{P}(\mathcal{M})$  is the probability of event  $\mathcal{M}$  coming true.  $F_x$  is the cumulative distribution in  $x$  with  $x \in [a, b]$  such that  $\int_a^b f_x(x) dx = 1$ . An equivalent expression can be proven in  $y$  giving  $F_y$ . For corresponding values of  $x$  and  $y$  it follows that

$$F_y(y) = F_x(x)$$

solving for  $y$  gives

$$y = g(x) = F_y^{-1}(F_x(x)).$$

A library written by Jan K. Trulsen (University of Oslo) in C++, with a numerical version of the Inversion method, was imported into the present code. The library includes the class *Inversion* with subroutine *draw*. These take as input the number of particles  $N$ , an arbitrary probability density function with boundaries, in one, two or three dimensions as input. The class with subroutines then draws  $N$  particles according to the specified probability density function. As an example, the probability density function of accelerated electrons,  $\hat{f}_{ea}(x, v)$ , taken from the present simulations is shown below:

---

```
//The PDF of accelerated electrons in a double layer
double fea(double x, double y){

    //Velocity y is drawn at the boundaries,
    //and transformed to it's velocity in position x
    double a=y*y+2.*QMe*phi(x);
    if(a*a<1e-8 && a < 0.0)a=0.0;
    double u=sqrt(a);

    //Maxwellian distribution
    //drifte declared elsewhere, and >0
    return exp(-.5*(u-drifte)*(u-drifte)/vthe/vthe);
}
```

---

### 4.4.2 The Box-Muller algorithm

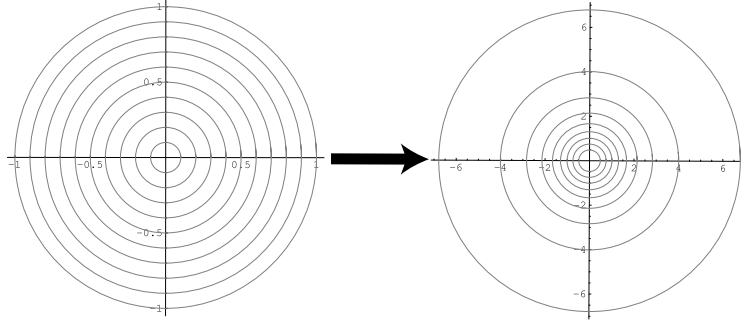


Figure 4.2: *Illustration of the Box-Muller transformation, from a uniform to a Gaussian random variable distribution.*

When transforming a uniform distribution  $f_x(x)$  to a normal distribution<sup>4</sup>  $f_y(y)$ , it is possible to arrive at an "analytic" solution of the Inversion method, called the Box-Muller algorithm [Truelsen, 2005]. It was first found by George Edward Pelham Box and Mervin Edgar Muller [Box and Muller, 1958]. The Box-Muller algorithm is as follows:

---

```
//Box Muller method:
//      Draws two random numbers from a Gaussian distribution
for (i<N){                                //For particle i
    do{
        //Draw x in U[-1,1]
        //Draw y in U[-1,1]
        //Calculate distance to origo squared
        rr=x^2+y^2;
    }while(rr>1.0); //If (x,y) inside circle of r=1
                    //end do-while loop

    //Calculate a
    a=sqrt(-2*log(rr)/rr);

    //Two random numbers are available from the above
    r1=y*a
    r2=x*a
}
```

---

The transform is illustrated in figure 4.2, where two uniform random numbers  $(x, y)$  are transformed into two Gaussian distributed random numbers  $(r1, r2)$ .

## 4.5 Simulation routines

The present simulations are performed with a C++ code, fully developed for this thesis by the author, with a few libraries included from other sources. All the codes developed by the author for the present thesis are found in appendix A.

The code use a one dimensional PIC method to perform the force calculations of each time step. The temporal evolution is performed with the Leapfrog orbital integration

---

<sup>4</sup>Equivalent to Maxwellian distributions, which is mentioned several places in this thesis.

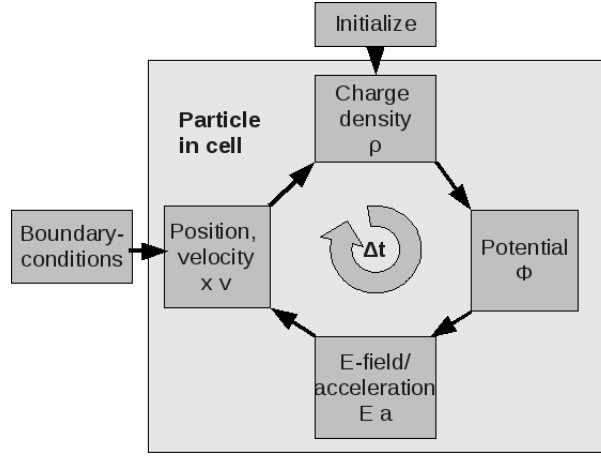


Figure 4.3: *Schematic overview of the operations done in a time step  $\Delta t$ . Initialization is (only) performed initially while boundary conditions are applied to all cycles.*

method, advancing the positions and velocities of the particles from the evaluated forces. The time cycle is illustrated in figure 4.3.

Particles are represented with two arrays per specie, one containing the particle positions and the other containing the velocities. Array cells of equal indices represent one particle. Initially, these arrays are filled with particles according to the prescribed probability density functions (PDFs), transformed with the Inversion method [Trulsen, 2005]. Particles leaving the simulation domain are removed from the arrays while influx particles are drawn, again according to the prescribed PDFs, and placed in the same arrays.

When in the time cycle the initialization and the insertion and removal of particles occurs is illustrated in figure 4.3.

In the remainder of this section, the methods calculating the charge density, potential, electric field, particle flux and performs the orbital integrations, are presented. Since it has experimented tested with three different methods for solving Poisson's equation, a presentation and comparison of these three is given in section 4.5.2. Based on these results, arguments are given for which method was chosen and why.

#### 4.5.1 The Particle-in-Cell method

As described in section 2.4.1, the PIC method distributes particles with continuous values for position onto a fixed spatial grid. Written in pseudo code the algorithm is as follows:

---

```

//The Particle-in-Cell method Cloud-in-Cell
//on a plasma defined between x=0 and x=L
//No. of gridpoints = Ncell
for(i<N){
    //Set spatial resolution
    dx=L/Ncell

    //Find position of particle i; x[i]
    x=x[i]

    //Find nearest lower and upper gridpoints
    X(j) = j*dx;

```

---

```

X(j+1) = (j+1)*dx;

// Calculate the weighting factor w
w = (x-X(j))/(X(j+1)-X(j));

// Distribute particles charge q to gridpoint j and j+1
Q(j)=(1-w)*q
Q(j+1)=w*q
}

```

---

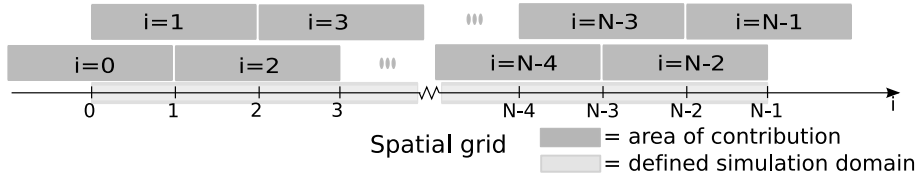


Figure 4.4: Illustration shows the area in which particles have weight contributions to a grid point  $i$  compared to the defined simulation domain.

The boundary grid points need a special treatment as the simulation domain only is defined between them and therefore the areas of contribution to these are half of other grid points, illustrated in figure 4.4. This can easily be dealt with, however it is done depends on whether the boundary conditions are open or periodic.

With periodic boundary conditions, the plasma defined in  $x \in [0, \ell]$  is assumed repeated in  $x \in [-\ell, 0)$ ,  $x \in [\ell, 2\ell)$  and so forth. The periodicity of the system implies that the area of contribution of the boundary grid points overlap and the particles assigned to each grid point can be summed into one, forgetting the other. As the upper boundary is not included in the simulation domain, the contents of the last grid point is moved in to the first, and the last excluded from further calculations, thus raising the charge density of the lower boundary grid point to the expected level.

With open boundaries, particles are allowed to enter and leave the simulation domain defined in  $x \in [0, \ell]$ . Assuming that the plasma density is constant within the area of contribution of a grid point, implies that the plasma assumed present outside the simulation domain will also contribute to the boundary grid points charge density. Thus multiplying the boundary grid points by 2, raises the level of the charge density to the expected level.

#### 4.5.2 Poisson's equation

Solving the discretized Poisson's equation (4.10) requires an iterative approach, which is solvable with a given right hand side of the equation and one boundary condition for each boundary (hence the name boundary value problem). For the system to be fully solvable, two boundary conditions (BCs) must be given to the system. These BCs can be of two forms, either Dirichlet or von Neumann. As presented earlier in section 2.4.4, Dirichlet type BCs keeps the value of the potential at a fixed value, i.e.  $\phi(0)$  or  $\phi(x = \ell)$  are known. von Neumann type BCs sets the derivatives of the potential in a likewise manner. Von Neumann BCs on the electrostatic potential of a plasma is equivalent to saying that the electric field is known, e.g.  $d\phi/dx = -E = E_0$ .

There are many different Poisson solvers. Three of them, Gauss-Seidel, Red-Black and Multigrid, are presented and compared in the following text.

### 1) Gauss-Seidel

The Gauss-Seidel method is the simplest, most straightforward solution to a boundary value problem. The discretized Poisson's equation (4.10) is solved by looping through all elements  $i$  from first to last, estimating a new approximation of  $\phi_i$  based on the previous approximation. This method requires an initial approximation for the first iteration, which should be chosen as close to the expected result as possible. The algorithm can be summarized as follows

---

```
//Gauss-Seidel solver of Poisson's equation
//Do iterations while the solution changes much
while(difference > tolerance){
    difference = 0.; //Reset difference
    for(i < Ncell){ //Loop through array
        //Save old approximation
        old = phi[i];

        //Solve the discretized Poisson's equation
        phi[i] = 0.5 * (rho[i] * dx^2 + phi[i+1] + phi[i-1])

        //Find square of difference
        difference += (phi[i] - old)^2;
    }

    //Special treatment of first and last element
    //of phi-array according to selected boundary
    //conditions (Dirichlet or von Neumann)
}
```

---

The tolerance sets the limit for when the solver has converged towards a solution, i.e. when the squared difference between the previous and current approximation is sufficiently small ( $\text{difference} < \text{tolerance}$ ). Typically, the difference will converge towards zero for increasing iterations and the tolerance is chosen just slightly larger.

The advantage of this algorithm is most importantly a simple implementation. If also the initial approximation is close to the final solution then this method can be relatively fast. This can be done by remembering the previous result and using it as the initial approximation in the next time step of the simulation. If, on the other hand, the initial approximation is far from the result, this method spends much time smoothing the solution rather than making coarse preliminary adjustments.

A clear disadvantage of the Gauss-Seidel is how this method is slow in propagating information of perturbations to the right hand side of the equation  $\rho$  to the solution  $\phi$ , as it loops through all elements successively in the array for every iteration.

### 2) Red-Black

The Red-Black has a Gauss-Seidel method in its core, but uses an alternative approach looping through the array elements when solving equation (4.10). The name of the method touches upon the essence of the method: Each element  $i$  in the array is "colored" red or black, as illustrated in figure 4.5. The red elements are calculated first, then the black.

Given a perturbation in the right hand side of the equation,  $\rho$ , the Red-Black method initially propagates this information across the entire array only on the "red" elements, hence in only half the time to that of Gauss-Seidel. With information of the perturbation already given in the "red" elements, the current approximation, which is found once the

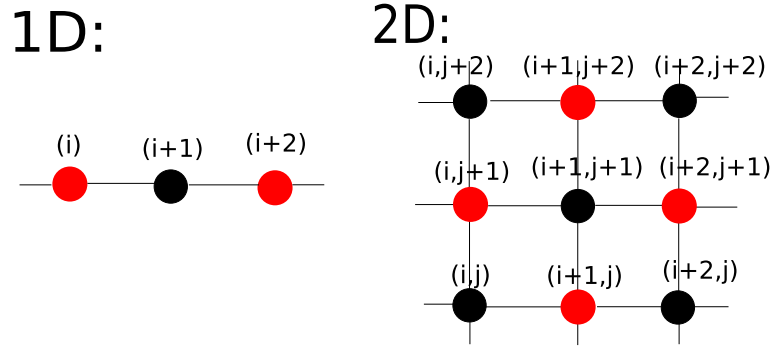


Figure 4.5: An illustration of how elements in a grid is "colored" in the Red-Black algorithm.

"black" elements also are looped over, is expected to be closer to the final solution than with Gauss-Seidel.

The Red-black method is therefore expected to be slightly faster and yield a more accurate result than Gauss-Seidel. In pseudo code, the Red-black algorithm is as follows:

---

```
//Red-Black solver of Poisson's equation
//Do iterations while the solution changes much
while(difference > tolerance){
    difference = 0.; //Reset difference
    for(i=even numbers < Ncell){ //Loop through red
        //Core of this loop as in Gauss-Seidel
    }
    for(i=odd numbers < Ncell){ //Loop through black
        //Core of this loop as in Gauss-Seidel
    }

    //Special treatment of first and last element
    //of phi-array according to selected boundary
    //conditions (Dirichlet or von Neumann)
}
```

---

### 3) Multigrid

The Multigrid method has been proven a fast solver for differential equations as it solves for solutions on coarser grids, working its way with solutions up through a hierarchy of grids to the finest grid available. Instead of doing small adjustments (smoothing) on the full grid size, as is done in the Gauss-Seidel method, Multigrid gains its speed by doing a large part of the workload on smaller grids. In present simulations, the Multigrid method does not take initial approximations as input, as is done in Gauss-Seidel and Red-Black.

The Multigrid method is illustrated in figure 4.6 consisting of four main operations:

- **Smoothing** - reducing high frequency errors, for example using a few iterations of the Gauss-Seidel method.
- **Restriction** - down-sampling the residual error to a coarser grid, while the accuracy found on a finer grid is communicated down through corrections done to the source charge density.



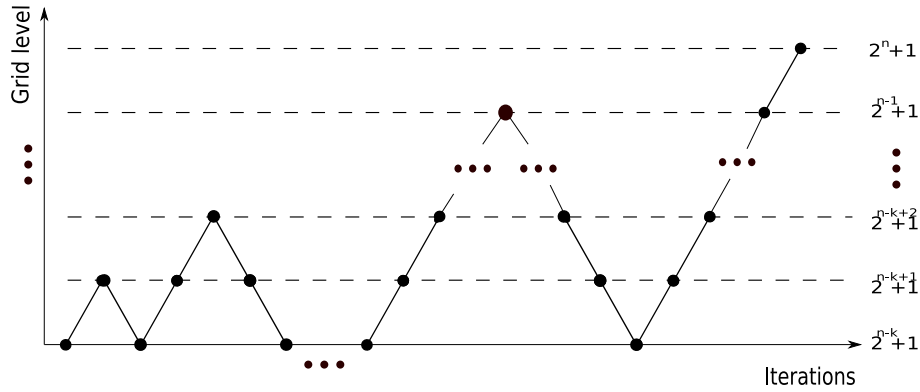


Figure 4.6: *The Multigrid way of iterating on courser grid before smoothing on the finest grid available. Ascending lines represent prolongation and descending lines restriction.*

- **Improving** - Computing corrections to the solution on a coarser grid.
- **Prolongation** - interpolating a correction computed on a coarser grid into a finer grid.

Assume that a charge density is given as an array of grid size  $N = 2^n + 1$ ,  $n = 1, 2, 3, \dots$ . The first approximation to the solution is found on a grid of size  $2^1 + 1 = 3$  or the lowest allowed from the program,  $2^{n-k} + 1$ ,  $k < n$  where  $k$  is set as the maximum Multigrid depth. The solution found on the coarsest grid is prolonged up to a higher grid before it is again restricted down onto the coarsest grid. By solving the equations on coarser grids, the shape of the final approximation is found faster than if all calculations were done on the finest grid, thus allowing a quicker convergence.

Between prolongation and restriction procedures, the method performs a given number of smoothing operations before and after the differential equation is solved at the localized finest grids. Localized finest grids can be recognized as the peaks of each triangle in figure 4.6. The Multigrid method continues until a set of prolongations reaches the grid size of the original charge density, and a final approximation to the solution is reached.

By simply counting the number of operations performed in a Multigrid method, it is possible to show that the calculations performed on the coarser grids, where much of the convergence rate is obtained, constitutes only a relatively small part of the total workload.

### A comparison of Poisson solvers

*Three Poisson solvers have been presented. Which is the best?*

1) Gauss-Seidel and 2) Red-Black operate only on the finest grid while 3) Multigrid considers a collection of different grid sizes. Hence, a direct comparison between the number of full iterations performed will not represent the same amount of workload on the simulation. However, counting the total number of cell operations in Multigrid<sup>5</sup> and dividing by the finest grid size will return an equivalent to a count of iterations in the Gauss-Seidel and Red-Black methods.

<sup>5</sup>Counting cell operations means counting every cell involved in prolongation, restriction, improving or smoothing processes, in addition to the solving of the differential equation at the selected grids.

$n$	$N$	$C$	$C/N$
2	5	28	7.6
3	9	81	12.1
4	17	188	14.7
5	33	401	16.0
6	65	824	16.7
7	129	1665	16.9
8	257	3340	17.0
9	513	6681	17.0
10	1025	13352	17.0
11	2049	26681	17.0

Table 4.1: *The Multigrid method's usage of computer power for different grid sizes.  $N = 2^n + 1$  : Grid size.  $C$  : Total number of cells calculated during one complete Multigrid calculation.  $C/N$  : Total operations on any cell to grid size ratio.*

Table 4.1 shows an overview of total number of cell operations  $C$  for a given grid size  $N$  in the Multigrid method. As an example, 3 smoothing procedures are performed before and after the solving of the differential equation on the localized finest grids (peaks of the triangles in figure 4.6). For each grid size  $N$ , the ratio  $C/N$  is calculated.

What is most striking of the results in table 4.1 is the asymptotic convergence of  $C/N$  to the number 17, that the workload of Multigrid for any grid size larger than 129 is comparable to a Red-Black or Gauss-Seidel method of 17 iterations.

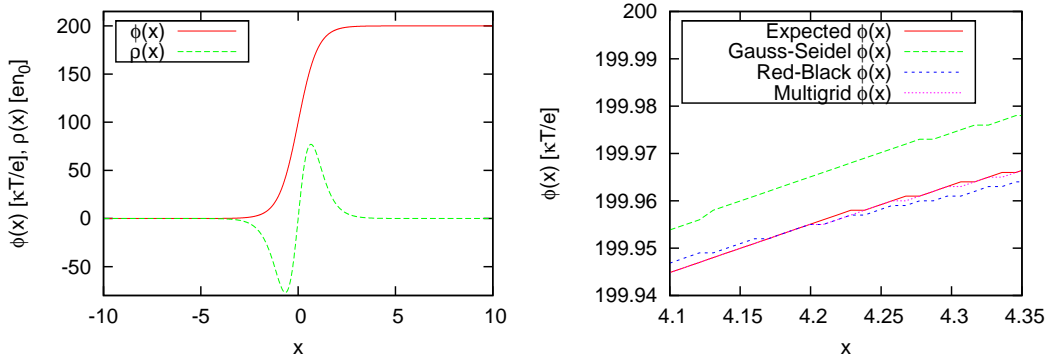


Figure 4.7: *Left panel: plot represents  $\phi(x)$  and  $\rho(x)$  as defined analytically, and thus the results expected given an ideal Poisson solver.  $\phi(x)$  also represents the perturbed initial approximations, and final results from all three solvers. The added perturbations and the variations between the final solutions of the solvers is not visible due to the scale of this plot. Right panel: the approximations of  $\phi(x)$  found from all three Poisson solvers (tolerance equals  $10^{-7}$ ) and the expected solution in.*

It is clear from table 4.1 that as long as the number of iterations performed by any of the other methods is significantly larger than 17, the Multigrid method will be the fastest. As speed is not the only quality of a good Poisson solver, the three have been directly compared by their accuracy, number of iterations and simulation time consumption.

Initially, a typical double layer potential,  $\phi(x)$ , equation (3.29), with corresponding charge density profile,  $\rho(x)$ , was assumed. The initial approximations of  $\phi(x)$  given to

the Gauss-Seidel and Red-Black Poisson solvers were set equal to equation (3.29), except the added relative noise level of amplitude  $\phi_{noise}/\phi_{DL} = 10^{-1}/10^2 = 10^{-3}$ . These initial approximations are supposed to mimic solutions found in previous time steps of the Gauss-Seidel and Red-Black methods. The initial approximation is plotted in the left panel of figure 4.7. As the Multigrid method does not take initial approximations, this had no effect whatsoever on its speed and accuracy.

Parameters in the Multigrid method were kept constant while the tolerance of Red-Black and Gauss-Seidel was varied. For every tolerance, solutions from the Poisson solvers were sought, calculating their final accuracy, simulation time and number of iterations. Accuracy is here defined as the absolute mean difference of the retrieved approximations to the expected solution. The potential  $\phi(x)$ , both the expected and approximations given by the three solvers, is plotted in figure 4.7 while comparisons of their accuracy, simulation time and iterations are made in figure 4.8.

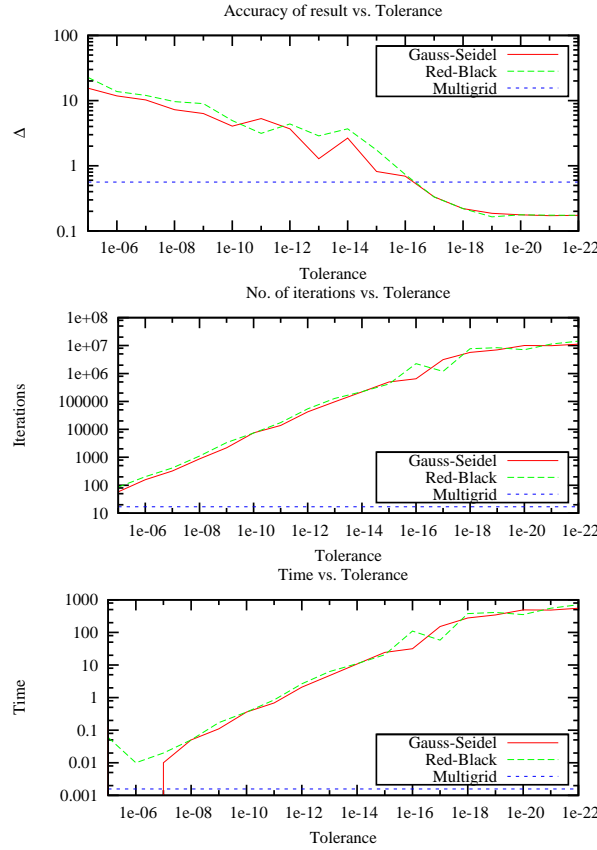


Figure 4.8: The x-axes show the set tolerance of Gauss-Seidel and Red-Black methods. Parameters of Multigrid is kept constant. Upper panel: The difference (absolute mean value) to the expected solution. Mid panel: Number of iterations. Lower panel: Time in seconds per. run.

Concluding from this figure there is little doubt that Multigrid is the better Poisson solver. The upper panel of figure 4.8, show that Red-Black and Gauss-Seidel reach the same level of accuracy when the tolerance is  $\sim 10^{-16}$ . With such low tolerance, both number of iterations and simulation time is extremely high,  $\sim 10^6$  and  $\sim 10^2$ s, compared to that of Multigrid ( $17$  and  $\sim 10^{-3}$ s).

There is also a general problem with the relative convergence criteria for Gauss-Seidel and Red-Black: With slow convergence, small successive relative errors can stop the procedure prematurely, although the absolute error may still be large.

*The Multigrid method was chosen as the Poisson solver for the present simulations.*

### 4.5.3 Electric field

Given an electrostatic potential, the electric field is found through equation (4.11). It is important to note that with this discretization of the first derivative, the E-field is calculated in the mid-gridpoints  $\dots, (i - 1/2), (i + 1/2), \dots$  compared to potential grid points  $\dots, (i - 1), i, (i + 1), \dots$  as well as the fact that numerical derivation in general amplifies numerical errors (noise).

The algorithm for calculating the electric field is given in pseudo code below:

---

```
//Electric field solver from known electrostatic potential
//Calculate dx, the equidistant spatial resolution between
// midpoint cells , dx=X(i+1)-X(i)
for (i<Ncell) E[i] = - (phi[i+1]-phi[i])/dx;
```

---

### 4.5.4 Equations of motion/Leapfrog algorithm

With the Leapfrog algorithm presented as discretized equations in section 4.3.1 the complete algorithm can be presented here in pseudo code. First, the initialization

---

```
//Backstepping the velocities , initializing the leapfrog algorithm.
for (i<N){
    //Converting v(t=0) to v(v=-dt/2)
    v[i] = v[i] - a[i]*dt/2
}
```

---

given the acceleration of particle  $i$ ,  $a[i]$  at  $t = 0$ , and then the Leapfrog algorithm

---

```
//Normal Leapfrog algorithm
for (i<N){
    //Stepping from velocity at t=t0-dt/2 to t=t0+dt/2
    // from a[i] taken at t=t0
    v[i] = v[i] + a[i]*dt

    //Stepping from position at t=t0 to t=t0+dt
    // from v[i] taken at t=t0+dt/2
    x[i] = x[i] + v[i]*dt
}
```

---

### 4.5.5 Flux density

Throughout the present simulations, time stationary solutions are sought. For all time stationary simulations, the flux of particles in and out of the system must stay constant and exactly cancel each other out. The opposite would impose a change in net charge of the simulated plasma.

Given periodic boundary conditions, particles leaving at the high end  $x = x_{out} > \ell$  re-enter at the low end with a new position  $x = x_{in} = x_{out} - \ell$ , and vice versa when leaving at the low end. Hence the net charge of the plasma never changes as no particles leave or enter the domain during the simulations.

Given open boundaries, particles are free to leave and enter the domain, and the flux of particles will be dependent on the probability density functions of the plasma. The number of particles entering or exiting the domain is  $\Delta N = P' \Delta t$ , where  $P'$  is given by the integral (4.6).

### Outflux

As the simulation deal with a single particle description, the flux of particles leaving the simulation domain is easily calculated. The program operates with lower and upper boundaries for the domain, here  $x \in [x_{lb}, x_{ub}] = [0, \ell]$ , so a test whether a particles position is no longer in the defined interval followed by the removal of the particle is sufficient. In terms of pseudo code, the algorithm becomes

---

```
//Removal of particles
for (i<N){
    //If particle x[i] no longer in [xlb,xub]=[0,L]
    if (x[i]>xub or xlb>x[i]){

        //Replace particle i with the last particle ,
        // indexed N-1
        x[i]=x[N-1];
        v[i]=x[N-1];

        //Reduce count of number of particles by 1
        N=N-1;
    }
}
```

---

The algorithm given above can be tested through comparing the experienced outflux with the expected flux of particles  $\Delta N$ . However this criterion is automatically fulfilled if the charge and particle density of the plasma is conserved. If it is found that during a simulation, particle or charge density is no longer conserved, a test as mentioned above would be in place.

### Influx

The number of particles entering at each boundary,  $\Delta N$ , was drawn with velocities following the distribution  $vf(v)$ . A numerical implementation of the Inversion method [Trulsen, 2005] was used to initialize the particles in the simulation according to  $vf(v)$ , as described in section 4.4.1. Corresponding positions were drawn with the algorithm:

---

```
//Setting positions of influx particles
// at boundary with position x=x0
for (i<deltaN){ \\\For all deltaN influx particles
    //Draw random number u from U[0,1]

    //Calculate position from influx velocity
    x[i]=x0 + u*v[i]*dt
}
```

---

## 4.6 Equilibrium plasma

The consistency of the routines presented earlier in this chapter were tested with simulations of a plasma configuration with a known solution; a plasma in thermal equilibrium

and homogeneous in space. Thermal equilibrium was achieved by introducing centered Maxwellian probability density functions for the entire plasma. Boundary conditions on the electrostatic potential for an equilibrium plasma is zero in all cases; Dirichlet or von Neumann on either of the two boundaries.

The initial probability density functions are as assumed in the above text, Maxwellian in velocity space and uniform in configuration space. The simplest way of initializing such a plasma would then be to draw velocities with the Box-Muller algorithm (see section 4.4.2) and positions directly from a uniform random number generator.

---

```
//Draw positions from a Uniform distribution , x in [0,L]
for (i<N) {
    //Find random number r in U[0,1]
    x[i]=r*L
}
```

---

Summarized, the following expectations must be met for the routines to be reliable:

- Charge density should be zero within the accuracy of the Particle-in-Cell routine.
- The electrostatic potential should be zero within the accuracy of the given Poisson solver and the noise level of the given charge density.
- Electric field should be zero within the accuracy of the discretization of the derivative and the noise level of the electrostatic potential.

If any of these expectations are not met, the plasma simulation algorithms are not consistent with plasma physics. *All routines proved to be reliable with equilibrium plasma simulations, satisfying the conditions above.*

## 4.7 Double layer

In this section, the double layer is introduced into the simulation algorithms presented earlier in this chapter. The double layer model implemented can be given the following general characteristics. The double layer potential profile is assumed to be monotonically *increasing* ( $d\phi/dx \geq 0$ ). The double layer is defined in  $x \in [0, \ell]$ , where  $\ell > 0$ ,  $\phi(0) = 0$  and  $\phi(\ell) = \phi_{DL} > 0$ . The accelerated electrons have drift velocities  $u_e > 0$ , and accelerated ions have  $u_i < 0$ . Both species  $s$  obey the relationship  $|u_s| = Uv_{ths}$ , where  $U$  represents the Mach number for each specie.

### 4.7.1 Initial probability density functions

When numerically initializing the probability density functions (PDFs) of a double layer, as in many other problems, there are numerous ways of reaching the designated goal. Before the present routines were implemented, attempts were made on two alternative methods. One was based on guessing the corresponding potential profile to the assumed probability density functions, and one based on the shooting algorithm. These procedures were discarded as they of different reasons, which will not be discussed here, was not sufficient for the present numerical model. In the final and working algorithm, PDFs are normalized so that particle densities can be adjusted with given forms of PDFs of entering particles.

### Weighting probability functions

Probability density functions (PDFs) were defined for all six populations in a double layer as in the example given in section 3.5.5. Given as functions of potential  $\phi$ , not coordinate space and with introduced scaling factors of the PDFs' amplitudes

$$\begin{aligned}
 f_{ea}(\phi_0, v) &= \alpha_{ea} \hat{f}_{ea, \phi_0}(v) = \alpha_{ea} \exp\left(-\frac{(v - u_e)^2}{2v_{the}^2}\right) \\
 f_{er}(\phi_{DL}, v) &= \alpha_{er} \hat{f}_{er, \phi_{DL}}(v) = \alpha_{er} \exp\left(-\frac{v^2}{2v_{the}^2}\right) \\
 f_{ed}(\phi_{DL}, v) &= \alpha_{ed} \hat{f}_{ed, \phi_{DL}}(v) = \alpha_{ed} \exp\left(-\frac{v^2}{2v_{the}^2}\right) \\
 f_{ia}(\phi_{DL}, v) &= \alpha_{ia} \hat{f}_{ia, \phi_{DL}}(v) = \alpha_{ia} \exp\left(-\frac{(v - u_i)^2}{2v_{thi}^2}\right) \\
 f_{ir}(\phi_0, v) &= \alpha_{ir} \hat{f}_{ir, \phi_0}(v) = \alpha_{ir} \exp\left(-\frac{v^2}{2v_{thi}^2}\right) \\
 f_{id}(\phi_0, v) &= \alpha_{id} \hat{f}_{id, \phi_0}(v) = \alpha_{id} \exp\left(-\frac{v^2}{2v_{thi}^2}\right).
 \end{aligned}$$

Note that the decelerated populations are reintroduced in this discussion. Even though the simulations performed in this thesis are performed on strong double layers, where decelerated populations do not exist, the initialization procedure is generalized for all both weak and strong double layers. Hence, the decelerated particle populations are included in the description of the double layer initialization.

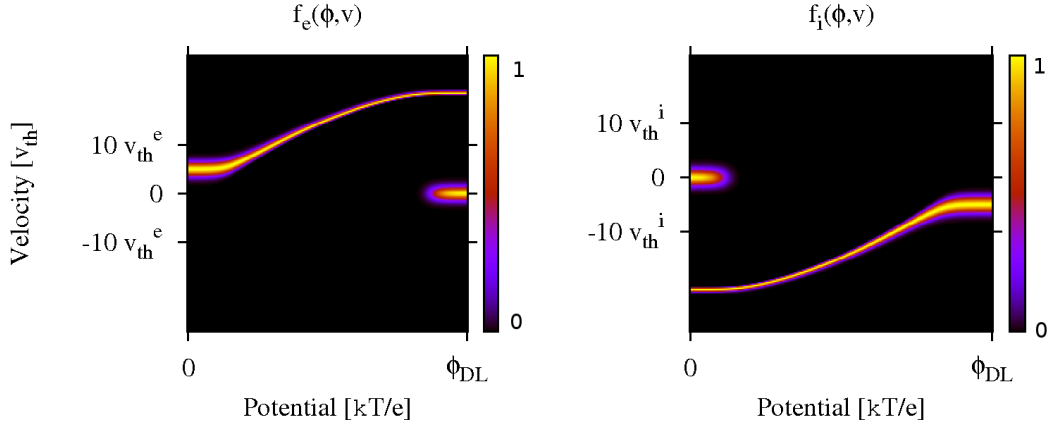


Figure 4.9: *Probability density functions of potential and velocity, as defined in the initialization of a double layer. The example shown here is of a strong double layer, as there are no decelerated populations present.*

The six unknown  $\alpha$ s can be found through six equations relating the PDFs. Three equations are derived from the charge density criteria,  $\rho(\phi = 0) = \rho(\phi = \phi_{DL}) = 0$  and  $\int_0^{\phi_{DL}} \rho(\phi) d\phi = V(\phi_{DL}) = 0$ . Applying the transformation of variables  $x \rightarrow \phi$  to equations (3.15) and (3.14), the probability density functions and particle densities

become

$$\begin{aligned} f_{sp}(\phi, v) &\equiv f_{sp, \phi_i}(v(\phi_i)) = \alpha_{sp} \hat{f}_{sp, \phi_i}(v(\phi_i)) \\ &= \alpha_{sp} \hat{f}_{sp, \phi_i} \left( \pm \sqrt{v^2 + \frac{2q_s}{m_s}(\phi - \phi_i)} \right) \\ n_{sp}(\phi) &= \int f_{sp}(\phi, v) dv = \alpha_{sp} \int \hat{f}_p^s(\phi, v) dv \equiv \alpha_{sp} \hat{n}_{sp}(\phi), \end{aligned}$$

and the charge density criteria can be expressed as

$$\begin{aligned} \alpha_{ia} \hat{n}_{ia}(0) + \alpha_{id} \hat{n}_{id}(0) + \alpha_{ir} \hat{n}_{ir}(0) - \alpha_{ea} \hat{n}_{ea}(0) - \alpha_{ed} \hat{n}_{ed}(0) &= 0 \\ \alpha_{ia} \hat{n}_{ia}(\phi_{DL}) + \alpha_{id} \hat{n}_{id}(\phi_{DL}) - \\ \alpha_{ea} \hat{n}_{ea}(\phi_{DL}) - \alpha_{ed} \hat{n}_{ed}(\phi_{DL}) - \alpha_{er} \hat{n}_{er}(\phi_{DL}) &= 0 \\ \alpha_{ia} N_{ia} + \alpha_{ir} N_{ir} + \alpha_{id} N_{id} - \alpha_{ea} N_{ea} - \alpha_{er} N_{er} - \alpha_{ed} N_{ed} &= 0 \end{aligned}$$

where  $\alpha_{sp} N_{sp}$  is the total number of particles of specie  $s$  and population  $p$ , defined by the integral  $\alpha_{sp} N_{sp} = \alpha_{sp} \int_0^{\phi_{DL}} \hat{n}_{sp}(\phi) d\phi$ .

Two more equations are found through the assumption of having no discontinuities in the PDFs of reflected and decelerated populations

$$\alpha_{er} = \alpha_{ed} \quad \text{and} \quad \alpha_{ir} = \alpha_{ed}.$$

This assumption is equivalent to saying that reflected and decelerated populations are products of the same plasma, which can then be described by one single probability density function.

The last  $\alpha$  is determined through equation (3.28) given in the section on the Sagdeev potential, section 3.5.4. This global  $\alpha$  raises the amplitude of the charge density, increasing the curvature of the double layer potential profile, so that for a given  $\ell$ , the charge density amplitude corresponds to the potential difference  $\phi_{DL}$ . The set of equations now matches the number of unknowns allowing the system to be solved and with corresponding amplitudes of the PDFs consistent with a double layer. The full set of PDFs as functions of potential and velocity is shown in figure 4.9.

The correctness of the solution hinges on the assumptions of PDFs for entering particles. If these assumptions are not physically correct, for instance not consistent with instabilities, then the solution is not correct.

### The potential profile

To find the full set of PDFs as functions of coordinate space, the potential profile  $\phi(x)$  must be found to complete the transform  $f(\phi, v) \rightarrow f(\phi(x), v)$ . Returning to the theory of the Sagdeev potential, to equations (3.27) and (3.24) a correlation between  $\rho(\phi)$  and a function  $x(\phi)$  is given, where  $\phi(x)$  is found through taking the inverse of  $x(\phi)$ . Normalized plots of  $\rho(\phi)$ ,  $V(\phi)$  and  $x(\phi)$  are plotted in figure 4.10, together with the resulting  $\phi(x)$ .

When initializing the potential profile with the above mentioned method, the double layer fills the entire simulation domain  $x \in [0, \ell]$ . As a consequence of this, the boundaries will restrict the system from acting freely and thus dominate its behaviour, giving no room for plasma instabilities to grow.

To allow the plasma to react more freely [Smith, 1982], the boundary plasmas where extended uniformly in space away from the double layer. With the extension, the double layer was still defined in  $x \in [0, \ell]$  while the plasma fills the range  $x \in [x_{lb}, x_{ub}]$  where  $x_{lb} < 0 < \ell < x_{ub}$ . Taking as an example  $[x_{lb}, x_{ub}] = [-\ell, 2\ell]$  the potential profile and corresponding PDFs are plotted in figure 4.11. The observed narrowing of the distribution functions of accelerated species, can be recognized as the phenomena called adiabatic cooling which was discussed in section 3.3.5.



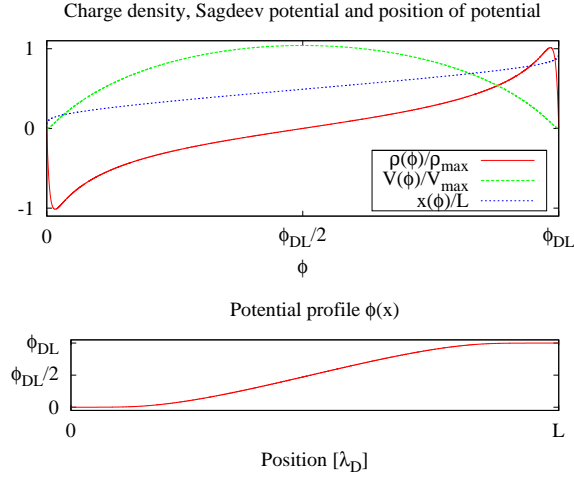


Figure 4.10: Upper panel: Normalized plots of charge density, Sagdeev potential and DL position coordinates (blue), all functions of potential  $\phi$ . Lower panel: The equivalent potential profile  $\phi(x)$ .

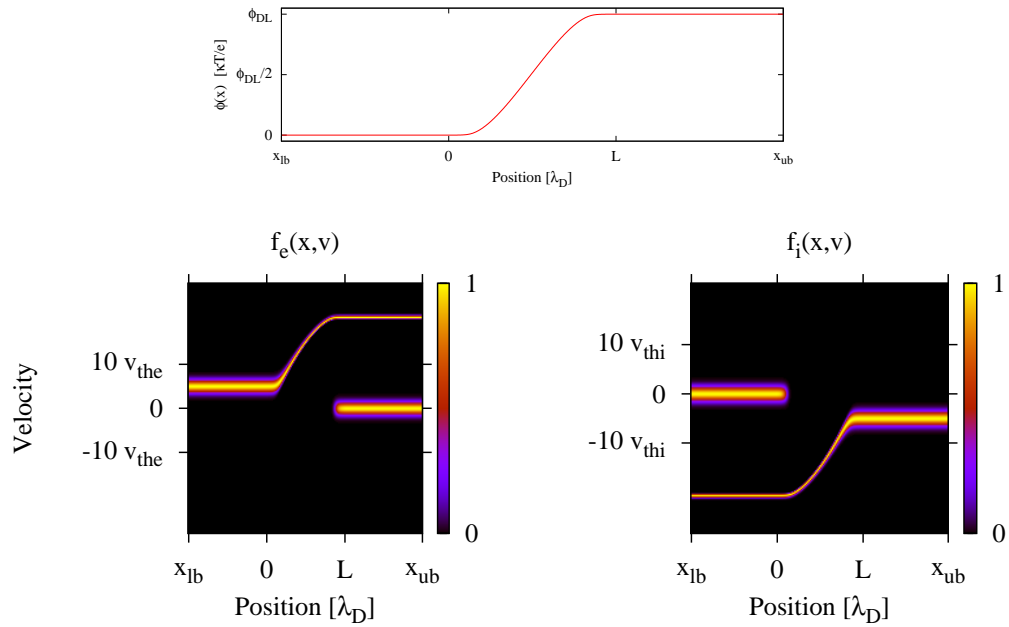


Figure 4.11: The electrostatic potential with the corresponding PDFs for a double layer, as initialized in present simulations.

## 4.8 Diagnostic routines

In this section, several diagnostic routines are presented, describing how they collect the data and how it is presented in figures. The routines were developed together with the plasma simulation routines presented in section 4.5. Some diagnostics routines were used frequently to determine the reliability of the numerical model. They are not included here as the general interest of this study lies not in confirming the stability of the code, but to present plasma physical effects. No simulations that appeared unreliable, judging from the mentioned diagnostic routines, were included as results in this thesis.

The diagnostic routines described here monitor the following properties of the plasma. Charge density, potential, electric field, particle density profiles, particle flux, phase space diagrams and harmonic oscillations through Fourier analysis.

The source code of the diagnostic routines are included in appendix A.

### 4.8.1 Charge density, potential and electric field

Charge density and the fields calculated from it (electrostatic potential and electric field) each represent the collective conditions of the simulated plasma at any time. Illustrating how these fields behave in temporal simulations thus gives an overview of the net plasma change, if any.

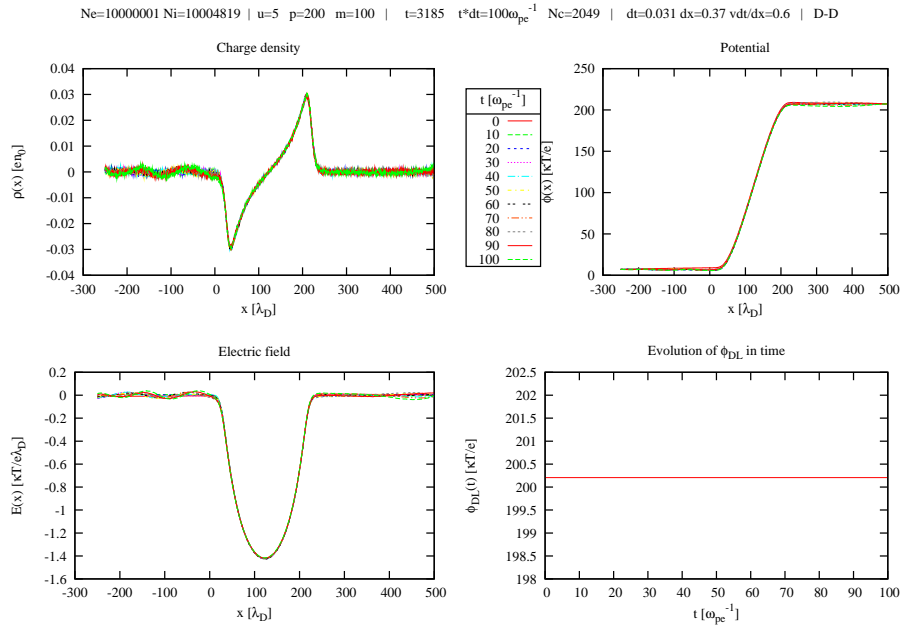


Figure 4.12: *First three panels: Stacked plots of charge density, electrostatic potential and electric field taken at 11 selected times of a simulation,  $t \in [0, 102\omega_{pe}^{-1}]$ . Last panel: The finite potential difference as a function of time.*

Diagnostics of the fields charge density, potential and electric field were performed with so called *field stack plots*. An example of such plots are given in figure 4.12. The first three panels show the mentioned fields for the initialized state, then over plotted by the same fields at 10 selected times during the simulations. The resulting plots are thus stacked plots of the respective fields at the selected times, hence the name. The last

panel of figure 4.12 illustrates the temporal variation in the potential difference over the double layer. As the example given here have D-D type boundary conditions, there is no variation in the potential difference whatsoever.

Stack plots prove efficient in monitoring whether the initialized state is stable, to some extent observe the properties of eventual disturbances and the level of simulation noise. A stack plot of a stable plasma is expected to appear as one graph, as the last plotted curve will cover the latter.

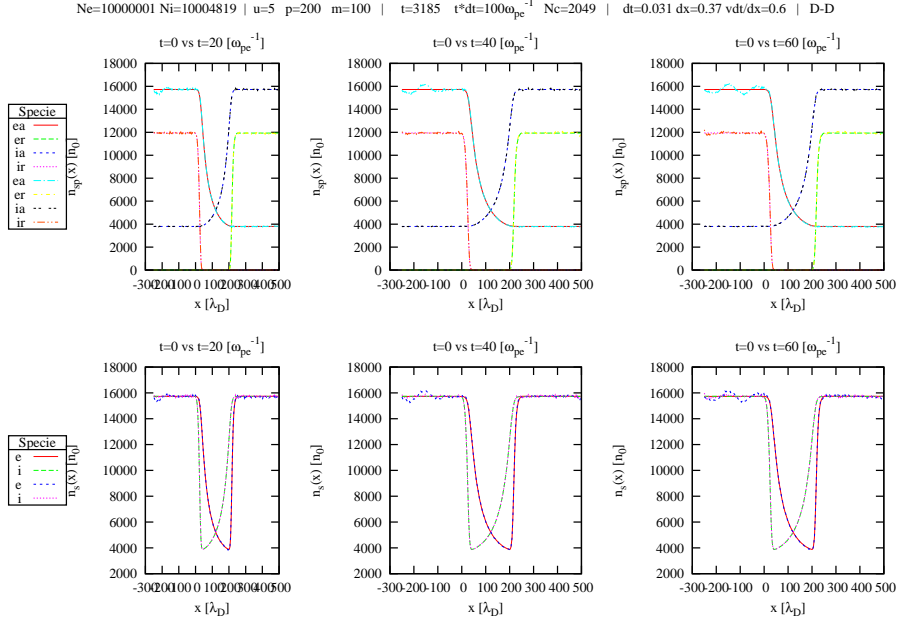


Figure 4.13: *Plots of density profiles of species (lower panels) populations (upper panels) taken at three selected times and overplotted with the density profiles of the initial state. The graphs representing the initial state are recognized as the four topmost entries in the upper and the two topmost entries in the lower legend.*

### 4.8.2 Density profiles

A more detailed study is done through separating the charge density into the particle density of each population and specie. As the two particle species are naturally separated in arrays, the separation of species requires no "tricks". Populations however, can be separated by their energy; i.e. a particle with velocity  $v$  and position  $x$  have  $mv^2 \leq |2q\phi(x)|$  the particle is trapped, free otherwise.

The resulting plots are density profiles of each specie and population. The density profiles are plotted for three selected times of the simulation together with the initial state. This way, any changes from the initial state will be seen as places where the curves no longer overlap. An example of such a figure is given in 4.13.

### 4.8.3 Flux

Flux of particles is an efficient way of monitoring conservation of the overall charge density of the plasma. Further, knowing the flux together with the bulk velocity of the

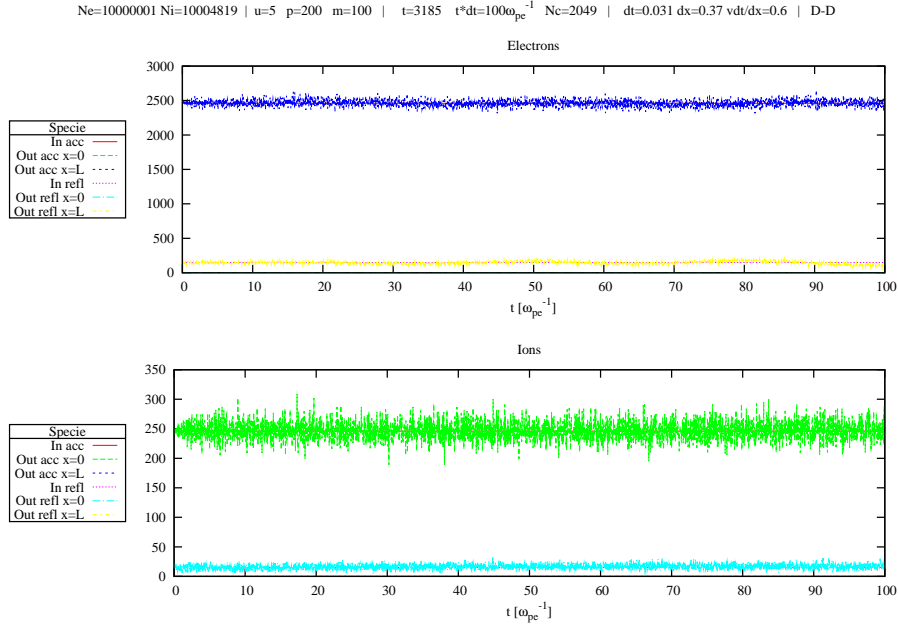


Figure 4.14: *Particle flux plotted as functions of time. The panels from top to bottom shows: 1. Flux of electrons, subdivided into free and trapped, 2. Flux of ions, subdivided into free and trapped.*

plasma the current density can be calculated and criteria like the Langmuir condition can be confirmed or disproved.

Flux is monitored by simply counting the number of particles either entering or exiting the simulation domain. Through sorting the particles into their respective populations before plotting, as done in previous section, plots like those shown in figure 4.14 can be retrieved.

#### 4.8.4 Phase space diagrams

Phase space diagrams are useful in illustrating the physics of many plasma physics phenomena like double layers, electron and ion holes. In present simulations the phase space diagrams allows a more detailed study of the effects observed in stacked field plots and density plots.

The phase space diagrams are made with a two dimensional histogram routine for each particle specie resulting in plots shown in figure 4.15. Temporal evolutions can be tracked through comparisons of diagrams taken at selected times.

#### 4.8.5 Fourier analysis

Fourier analysis is useful in identifying oscillations modes in physical phenomena. In present simulations, a spatial Fourier analysis was implemented to identify the periodicity of perturbations, if any. Dominating oscillation modes identified as peaks in the power spectrum, were tracked throughout the simulations resulting in a plot of oscillation amplitude versus simulation time, as seen in figure 4.19. The full procedure used to arrive at plots like figure 4.19 is described in the following text. The studied regions

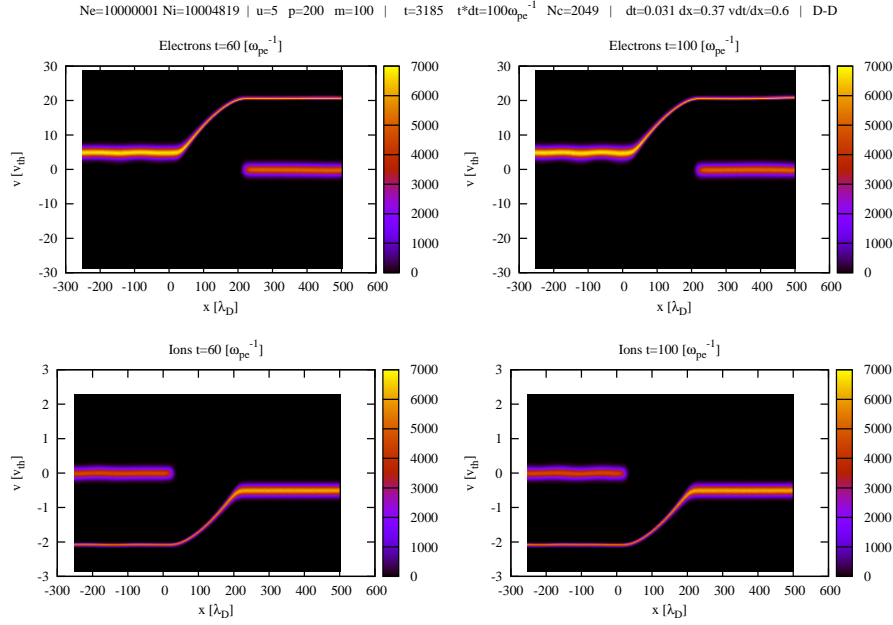


Figure 4.15: *Phase space diagrams of electrons (first row) and ions (second row) at  $t = 61.34\omega_{pe}^{-1}$  (left) and  $t = 102.35\omega_{pe}^{-1}$  (right).*

of the simulation domain was the initially homogeneous plasmas outside of the double layer, each of width  $250\lambda_{De}$ .

The Fourier analysis was performed with a Fast Fourier Transform (FFT). FFT requires a grid size of  $N = 2^n$  for  $n = 1, 2, 3, \dots$  for the data to be studied and likewise returns results in arrays of the same dimension. The value of  $N$  is selected as the highest possible fulfilling the criterion  $N \cdot dx < 250\lambda_{De}$ . The spatial resolution  $dx$  can be found from the relation  $dx = L_{sys}/(N_{cell} - 1)$ , where the full grid size  $N_{cell} = 2^m + 1$ , for  $m = 1, 2, 3, \dots$ , is chosen to be the smallest value fulfilling the condition  $dx < 0.5$ . From the above given method it is obvious that  $L_{FFT}$  depends on the system length through  $dx$ .

FFT returns the power spectra as an array, where the array indices  $i = 0, 1, 2, 3, \dots$  represents the DC-level (direct current), first, second, third, etc. harmonics, respectively. The first harmonic is the lowest detectable wave number  $k_1 = 2\pi/L_{FFT}$  by the FFT. Second, third, etc. harmonics have wave numbers  $k_i = 2\pi i/L_{FFT}$ . Any wave with  $k < k_1$  (and therefore wavelength  $\lambda > L_{FFT}$ ) will be regarded as a DC-level by the FFT.

Before performing the Fourier analysis, the window function, illustrated in figure 4.16, was multiplied to the selected regions, thus removing any discontinuities at the boundaries<sup>6</sup>. Examples of such a selection is given in figure 4.17. The power spectrum of the windowed sections was found with a FFT at every time step and accumulated over a time interval to reduce relative noise. The length of these intervals are chosen to be 10% of the full simulation length. Peaks in the power spectrum were identified from the following criteria:

- If the derivative of the power spectrum changes sign from positive to negative a peak is identified.

<sup>6</sup>Process referred to as "windowing".

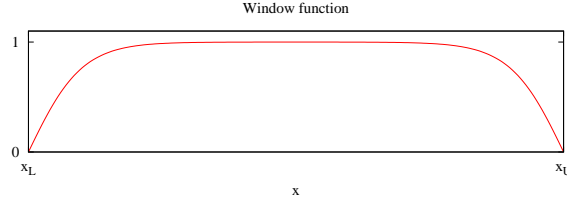


Figure 4.16: *The window function multiplied to a one dimensional field subject to Fourier analysis. In present simulations, this window is applied twice, once on each side of the double layer.*

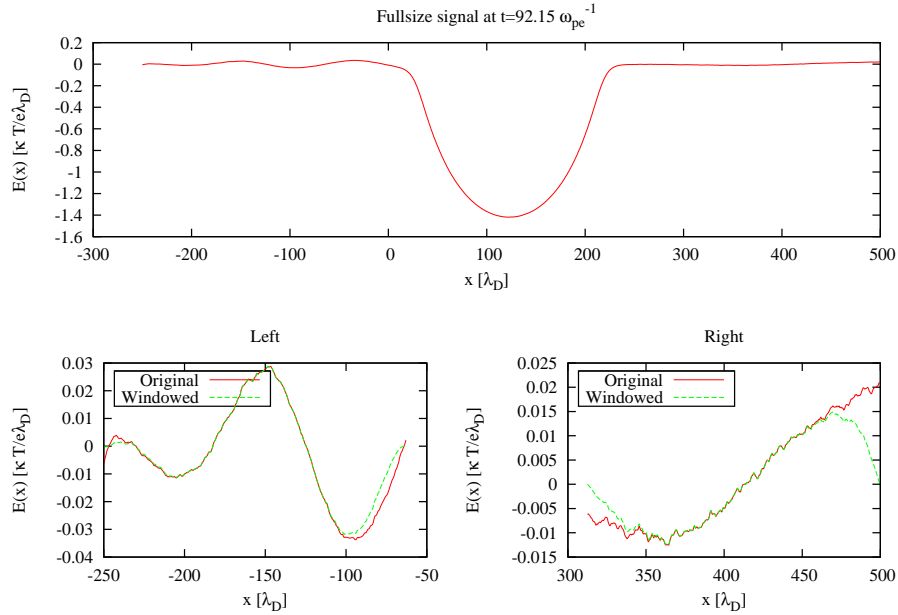


Figure 4.17: *Upper panel: the original electric field signal, Lower panels: The signal separated in left and right, before and after multiplication with the window function.*

- By setting a tolerance level  $t = \beta m$  where  $m$  is the mean value of the spectrum and  $\beta$  a proportionality constant, all peaks with power lower than  $t$  are ignored.

The power spectrum of the example field given in figure 4.17, can be seen in figure 4.18 plotted together with the mean and tolerance values discussed above. Through these criteria, only the peaks in the power spectrum which grow large compared to the mean value of the spectrum are studied further.

The identified oscillations are remembered throughout the simulation, and for every complete accumulation of power spectra, the power of each mode and it's development through time is stored. This data is plotted as a function of time giving plots like those in figure 4.19. In the present example, the algorithm did not identify any peaks in the power spectra on the right hand side. Thus, no plot of the right hand side is presented in figure 4.19.

It is expected that once an oscillation mode of wavenumber  $k$  grow dominant, the

second harmonic with wavenumber  $2k$  and combinations of these two grow together with the first. An example of such an effect is seen in figure 4.19, and the study of these may give important clues to how the plasma behave.

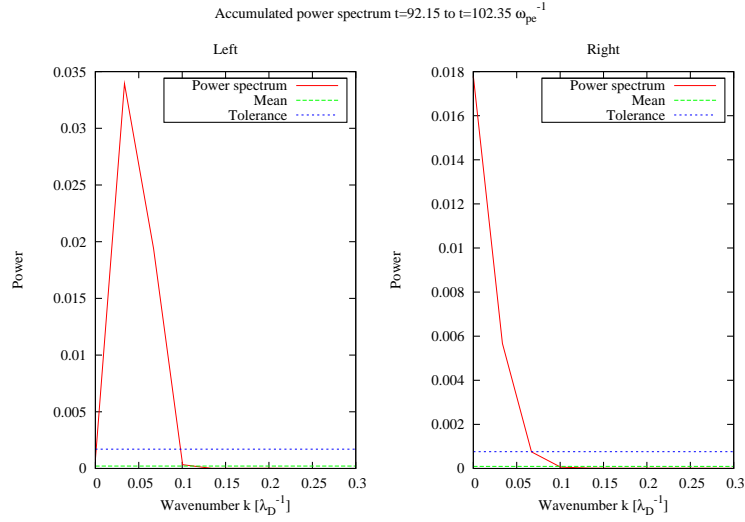


Figure 4.18: *The power spectrum of both sections at the last accumulation of power spectra of the simulation time. The horizontal lines in both panels are the mean and tolerance levels of both power spectra.*

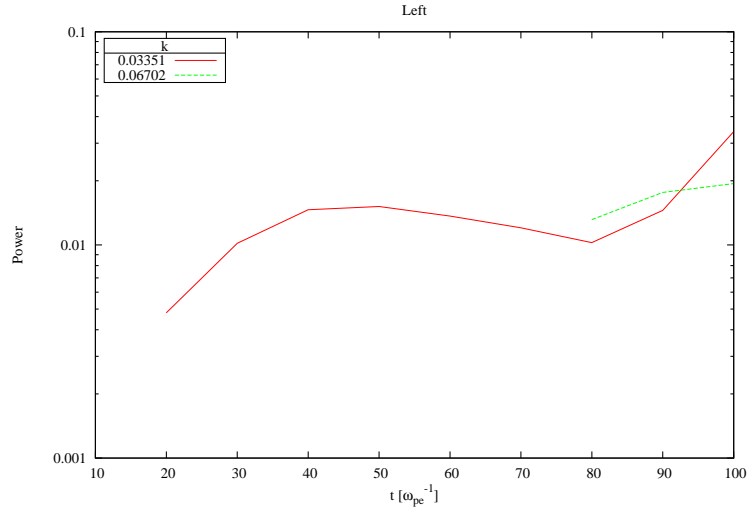


Figure 4.19: *Tracking of the dominating oscillations as a function of time. On the vertical axis is the power of the oscillation.*





# Chapter 5

## Results

The goal of the present study is to test under which conditions double layers are able to exist. The study is performed with the numerical model introduced in chapter 4 and the results presented here are illustrated with the type of figures discussed in section 4.8.

Typical parameters describing the plasma double layer and boundary conditions of the numerical model are varied, investigating their effect on the results. For all simulations the stability, frequency of harmonic oscillations and the effect of two stream instabilities are studied.

Parameter	Symbol	Range	Units
Double layer length	$L$	[50,750]	$\lambda_{De}$
Double layer strength	$\phi_{DL}$	[25,800]	$\kappa T/e$
Mass ratio	$m_i/m_e$	[1,300]	
Drift velocities	$U$	[1.8,9]	$v_{the}$

Table 5.1: *List of the variable parameters of the present simulations. Here shown with given symbol, defined ranges in which they are varied and their units.*

The variable parameters, with their given ranges of values and units, are shown in table 5.1. The notation applied to the boundary conditions (BCs) are as defined earlier: "left BC"- "right BC" and with the abbreviations D and vN for Dirichlet and von Neumann boundary conditions types.

Table 5.3 (on page 82) contains an overview of the parameter combinations used in some of the present simulations, marked with a filled circle (●) or a number referring to a figure in the text. Each simulation was run four times, one for each boundary condition combination as mentioned earlier. Where the analysis required so, some additional simulations to those listed in table 5.3 were performed.

### 5.1 Reference simulation

The simulation with  $L = 250$ ,  $\phi_{DL} = 200$ ,  $m_i/m_e = 100$  and  $U = 5$  was chosen as a reference simulation, since its parameters are close to the center of each parameter's range (see table 5.1). The results of all simulations are thus compared to the results of the reference simulation, creating an overview of the results variations with the changes of the four mentioned parameters.

Field stack plots of the reference simulation are shown in figure 5.1 for all four combinations of boundary conditions. The remaining plots of the forms presented in section 4.8, are shown in figures 5.2 and 5.3 for the D-vN boundary condition combination.

Within the simulation time of  $t = 100\omega_{pe}^{-1} = 10\omega_{pi}^{-1}$  the relative amplitude of perturbations, seen in charge density, electric field and potential, remains small (see figure 5.1) for D-D and vN-D boundary conditions. D-vN and vN-vN boundary conditions on the other hand induce higher relative amplitudes, up to and above what is regarded as unstable. Thus, for conditions represented by the reference simulation, a double layer is more likely to exist for D-D and vN-D boundary conditions.

Figure 5.2a, illustrating the temporal evolution of dominating peaks in the power spectrum (taken outside the double layer), shows how respectively three and one peaks in the spectrum of the left and right hand side plasmas evolve with time. In the left hand side plasma, the initial wave of  $k = k_1 = 0.03351\lambda_{De}^{-1}$  experiences increasing amplitude up to about  $t = 80\omega_{pe}^{-1}$  and decreasing afterwards. The  $k = 3k_1 = 0.1005\lambda_{De}^{-1}$  wave stays roughly constant in amplitude up until the same moment. At  $t = 90\omega_{pe}^{-1}$ , the wave with  $k = 2k_1 = 0.06702\lambda_{De}^{-1}$  appear and immediately increases together with the  $k = 3k_1$  wave. The right hand side plasma, the only wave present have  $k = 3k_1 = 0.1005\lambda_{De}^{-1}$ . How the values of  $k$  relate to the Buneman instability will be discussed in section 5.3.2.

The density profiles of figure 5.2b show that the perturbations seen in previous figures are primarily in the accelerated electron density. They have clear harmonic oscillation tendencies and appear close to the boundary at the low potential side of the double layer. In the later image ( $t = 60\omega_{pe}^{-1}$ ) reflected ions have become slightly perturbed as well, as a reaction to the electron oscillations.

Both in- and outflux of each particle population appear equal and constant in time up to  $t \approx 70\omega_{pe}^{-1}$  judging from figure 5.3a. Fluctuations are visible only for accelerated electrons leaving the simulation domain.

Phase space diagrams, seen in figure 5.3b, reveal the effect of the above mentioned oscillations on the velocities of accelerated electrons and reflected ions. These diagrams are taken at a few selected times,  $t = 60\omega_{pe}^{-1}$  and  $t = 100\omega_{pe}^{-1}$ . At  $t = 100\omega_{pe}^{-1}$ , the perturbations in velocity  $\Delta v_s$  for the accelerated electrons ( $s = ae$ ) and reflected ions ( $s = ri$ ) relate to perturbations in the accelerated electron density  $\Delta n_{ae}$  such that

$$\begin{aligned} \Delta v_{ae} > 0 \quad \text{and} \quad \Delta v_{ri} < 0 \quad \text{for} \quad \Delta n_{ae} < 0 \\ \Delta v_{ae} < 0 \quad \text{and} \quad \Delta v_{ri} > 0 \quad \text{for} \quad \Delta n_{ae} > 0 . \end{aligned}$$

The reference simulation was once run with higher particle numbers, increasing from  $10^7$  to  $10^9$ . The only observed effect on the results was the expected decrease of noise by a factor of  $\sqrt{10^9/10^7} = 10$ .

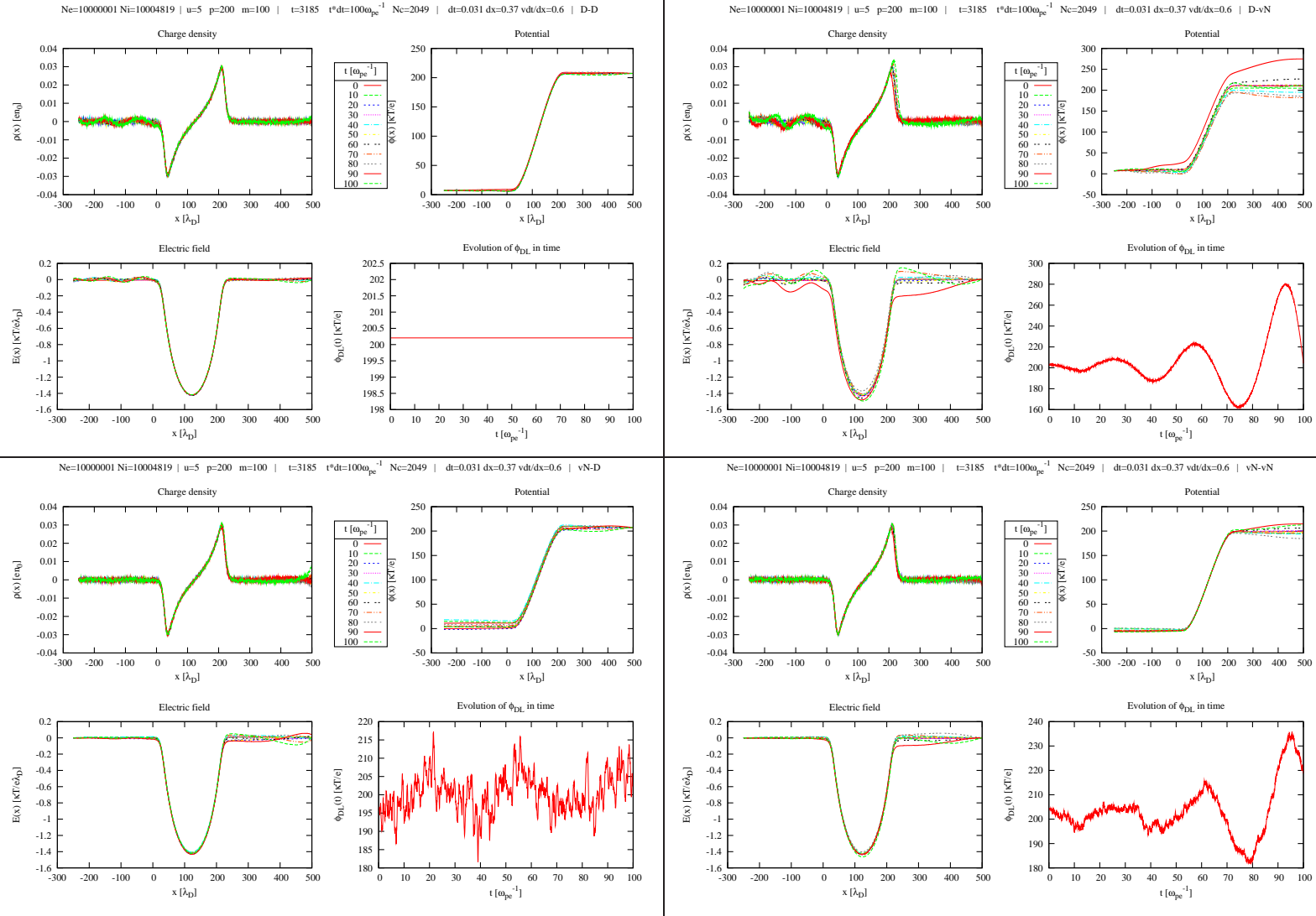
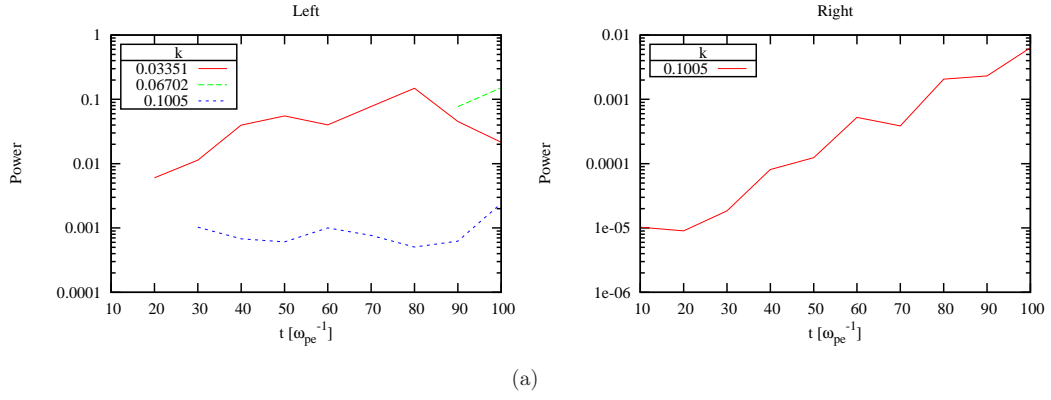


Figure 5.1: *Field stack plots for  $L = 250\lambda_{De}$ ,  $\phi_{DL} = 200$ ,  $m_i/m_e = 100$  and  $U = 5$ , defined as the reference run. The four sets represent different choices of boundary conditions, D-D, D-vN, vN-D and vN-vN reading from top left towards bottom right.*



Ne=10000001 Ni=10004819 | u=5 p=200 m=100 | t=3185 t\*dt=100ω<sub>pe</sub><sup>-1</sup> Nc=2049 | dt=0.031 dx=0.37 vdt/dx=0.6 | D-vN

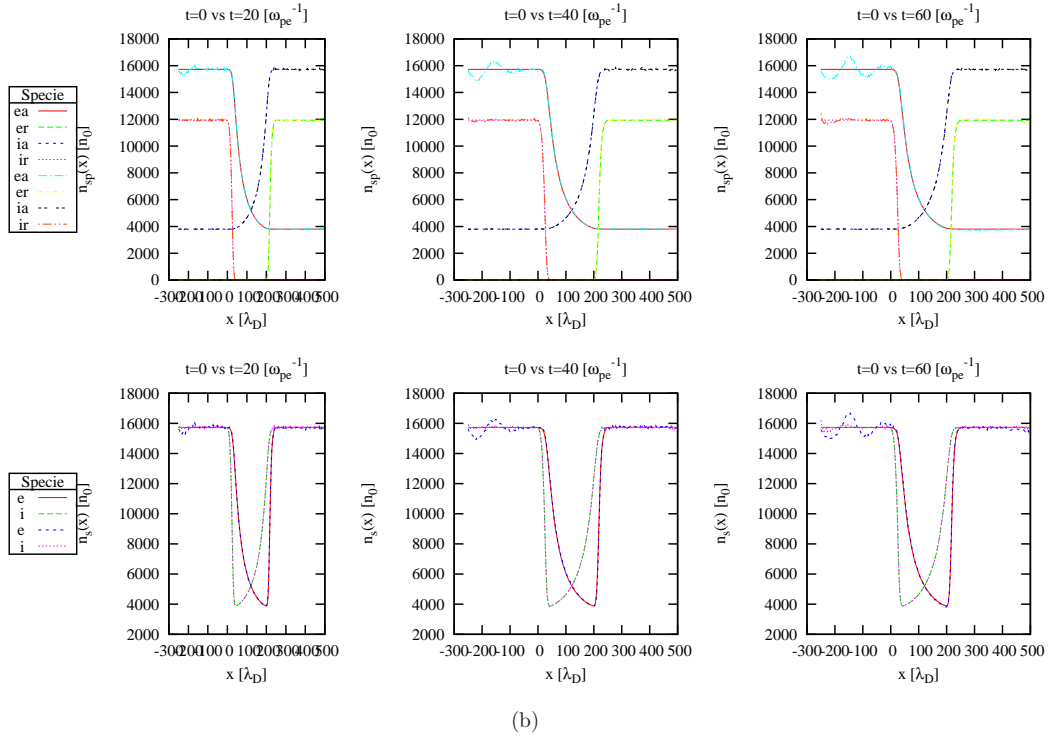


Figure 5.2: *Reference run for the D-vN boundary conditions. From top to bottom: tracing of dominating peaks in the power spectrum and density profiles.*

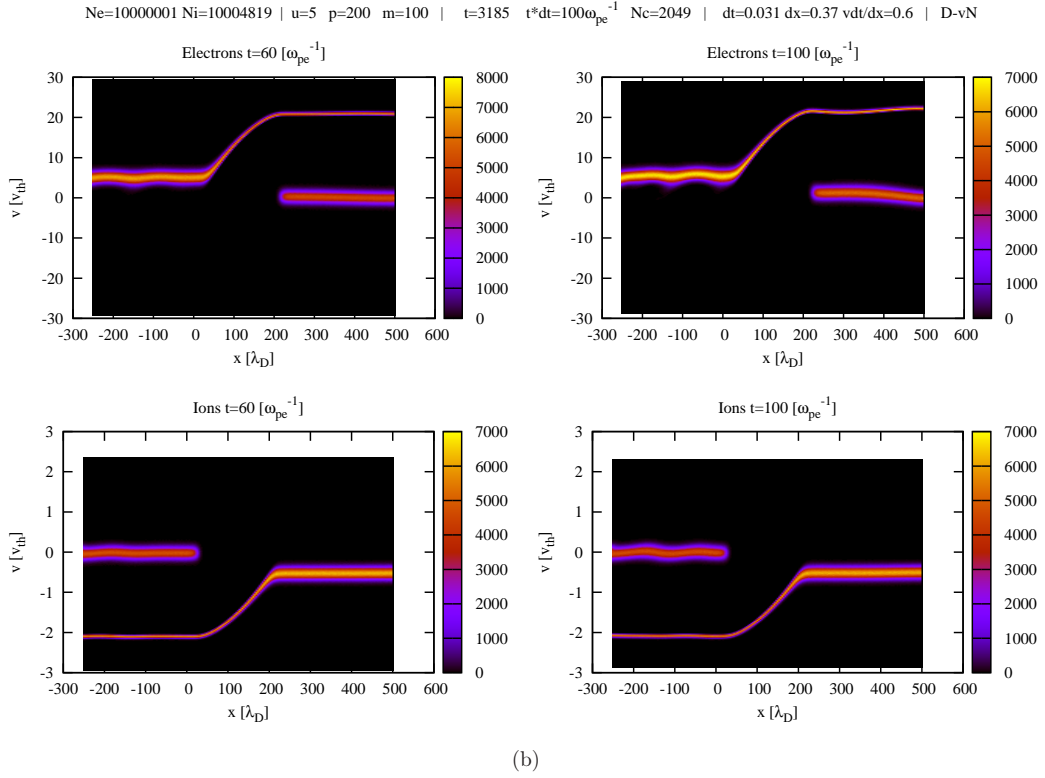
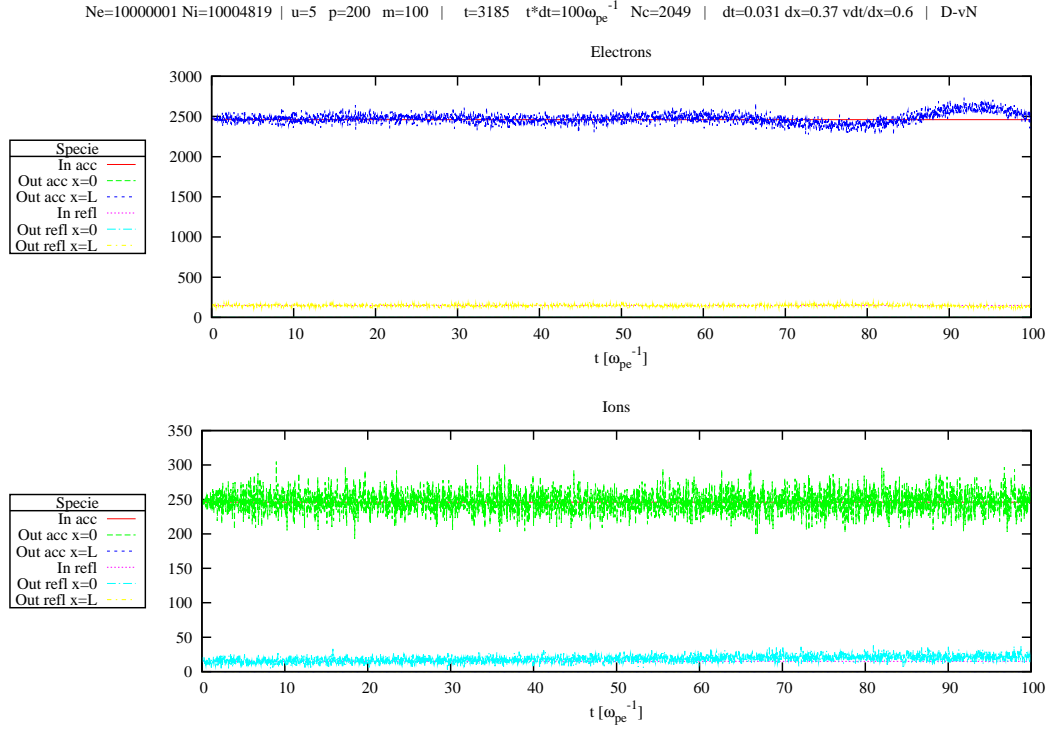


Figure 5.3: Reference run for the D-vN boundary conditions. From top to bottom: Particle flux and phase space diagrams.

## 5.2 Stability

Through studying the stability of the simulations, it is possible to determine whether the plasma condition, defined by the chosen parameters and boundary conditions, supports the existence of a double layer. In the present studies, the stability of a simulation is defined in the following way:

If the maximum value of any perturbation in either of the fields describing the double layer (charge density, potential and electric field) grows larger than 50% of the initial maximum value of the respective field, the simulation is said to have grown unstable. The time when this happens (measured in electron plasma periods), hereafter defined the simulation's *lifetime*  $\tau$ , is thus a measure of the simulation's *stability*. Note that this is not a numerical but a physical measure of stability.

Investigations of the power spectra of the plasmas outside the double layer can be used as a confirmation to the definition given above. An unstable simulation is expected to coincide with an increase of the power spectra, both in dominating peaks and a general noise level.

In this section, the effect the boundary conditions and chosen parameters have on the stability is investigated. Finally, the results from present simulations are compared to the scaling law, which was introduced in section 3.3.1.

### 5.2.1 Boundary conditions dependencies

Boundary conditions are intended to mimic real physics, e.g. the current generators discussed in section 2.4.4. However, the mathematical equivalents of realistic boundary conditions may not be quite correct. In which case, boundary conditions may introduce effects which can not be regarded as physical properties of the simulated plasma.

In present simulations, four different combinations of boundary conditions were used for each simulation. Comparing differences of the four boundary condition combinations, allows for a detailed study of effects which can be caused by the respective boundary conditions.

From the stacked field plots of the simulations  $L = 750$ ,  $\phi_{DL} = 50$ ,  $m_i/m_e = 100$  and  $U = 5$  (figure 5.4) and  $L = 750$ ,  $\phi_{DL} = 500$ ,  $m_i/m_e = 100$  and  $U = 5$  (figure 5.5) it is found that D-D induces low amplitude perturbations for  $\phi_{DL} = 50$  and high amplitudes for  $\phi_{DL} = 500$ . The opposite is observed for vN-vN boundary conditions. D-vN is found to be less stable for high  $\phi_{DL}$  and the large amplitude perturbation mainly arise on the left hand side of the double layer. vN-D is generally more stable for high  $\phi_{DL}$ , and as will be discussed in section 5.3, is the only boundary condition combination where damped oscillations occur.

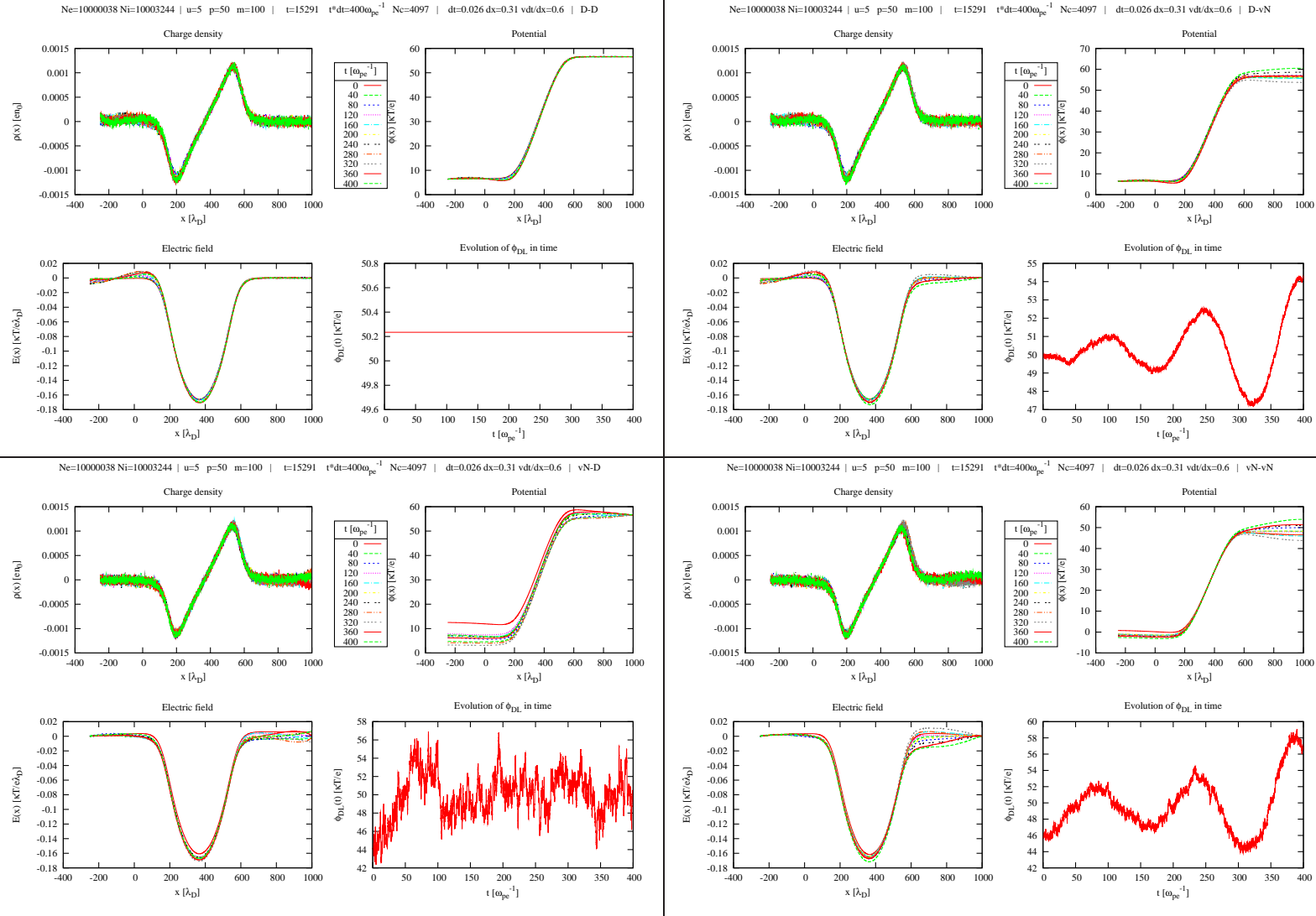


Figure 5.4: *Field stack plots for  $L = 750\lambda_{De}$ ,  $\phi_{DL} = 50$ ,  $m_i/m_e = 100$  and  $U = 5$ . The four sets represent different choices of boundary conditions, D-D, D-vN, vN-D and vN-vN reading from top left towards bottom right.*

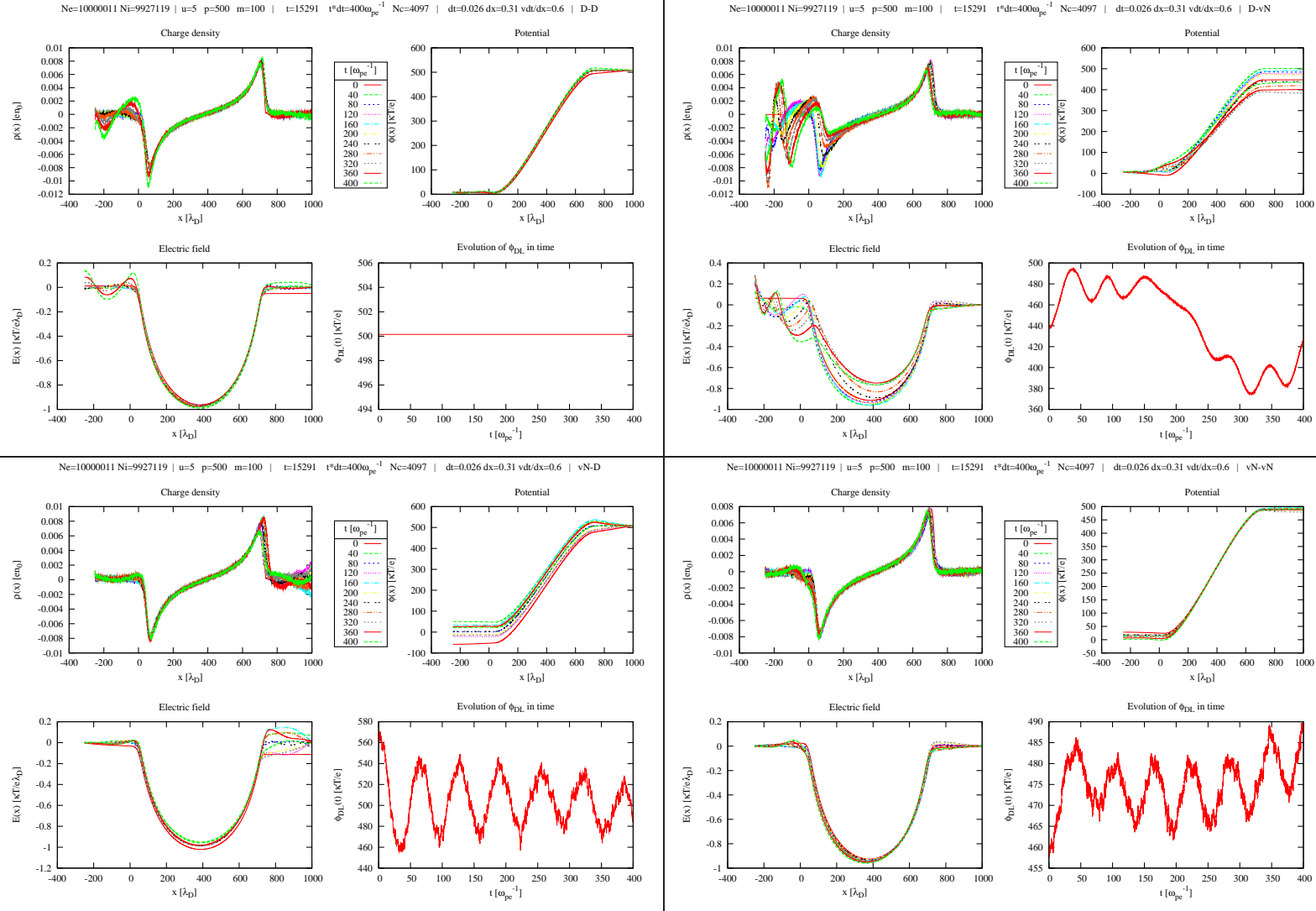


Figure 5.5: Field stack plots for  $L = 750\lambda_{De}$ ,  $\phi_{DL} = 500$ ,  $m_i/m_e = 100$  and  $U = 5$ . The four sets represent different choices of boundary conditions, D-D, D-vN, vN-D and vN-vN reading from top left towards bottom right.



### 5.2.2 Parameter dependencies

The primary stability investigation is done on the general variation of lifetime with the parameters listed in table 5.1. While details of such variations are given below, they can be summarized as follows:

- $L$  - Increasing lifetime for increasing  $L$ .
- $\phi_{DL}$  - For  $L \lesssim 250$ , simulations with  $\phi_{DL} \lesssim 300$  tend to be more stable. For  $L > 250$  there are no clear variations in lifetime with  $\phi_{DL}$ .
- $m_i/m_e$  - Strong dependency for mass ratios close to unity, weak dependency otherwise ( $m_i/m_e \gg 1$ ).
- $U$  - Increasing lifetime with increasing  $U$ .

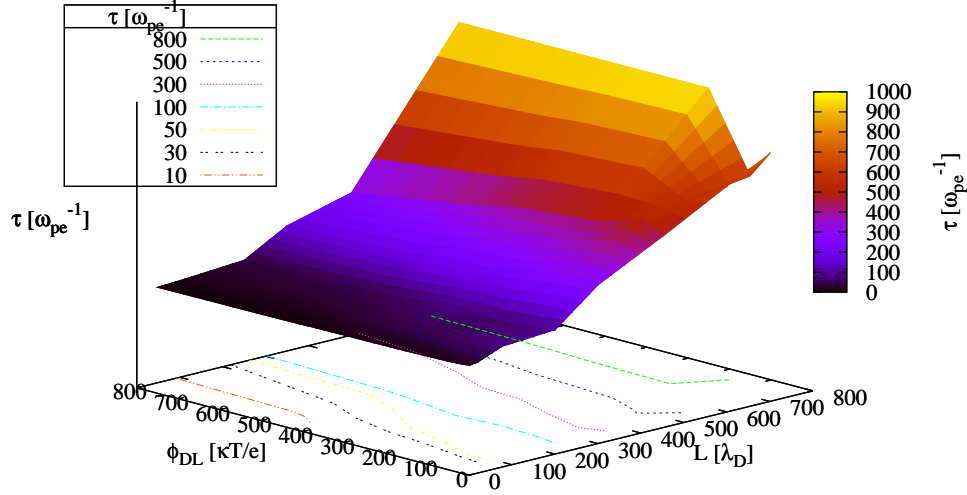


Figure 5.6: *Lifetime as a function of  $L$  and  $\phi_{DL}$ , averaged over velocity. Mass ratio is  $m_i/m_e = 100$ . Contours of the surface plot is drawn at the base of the figure.*

The stability variations with  $L$  and  $\phi_{DL}$  are demonstrated with figure 5.6 where *lifetime*  $\tau$  is given as a function of  $L$  and  $\phi_{DL}$  (for  $m_i/m_e = 100$ , averaged over  $U$  and boundary conditions). Isolating the single variable dependency of  $\tau(L, \phi_{DL})$  reveals that

$$\tau(L, \phi_{DL}) \sim L \quad \text{for } \phi_{DL} \in [50, 800] , \quad (5.1)$$

$$\tau(L, \phi_{DL}) \sim -\phi_{DL}^2 \quad \text{for } L \in [50, 250] , \quad (5.2)$$

$$\tau(L, \phi_{DL}) \sim \phi_{DL} \quad \text{for } L \in [350, 750] . \quad (5.3)$$

In general, the mass ratio was set  $m_i/m_e = 100$  for all combinations of  $L$ ,  $\phi_{DL}$  and  $U$ . For simulations where  $\phi_{DL} = 200$  and  $U = 5$ , mass ratios of  $m_i/m_e = 1$  and  $m_i/m_e = 300$  were also investigated. The unity mass ratio proved to be highly unstable as the ions no longer contained higher inertia than electrons and thus the plasma responded quicker to perturbations. Mass ratio of 300 showed little change in the results except a slight increase of ion response time.

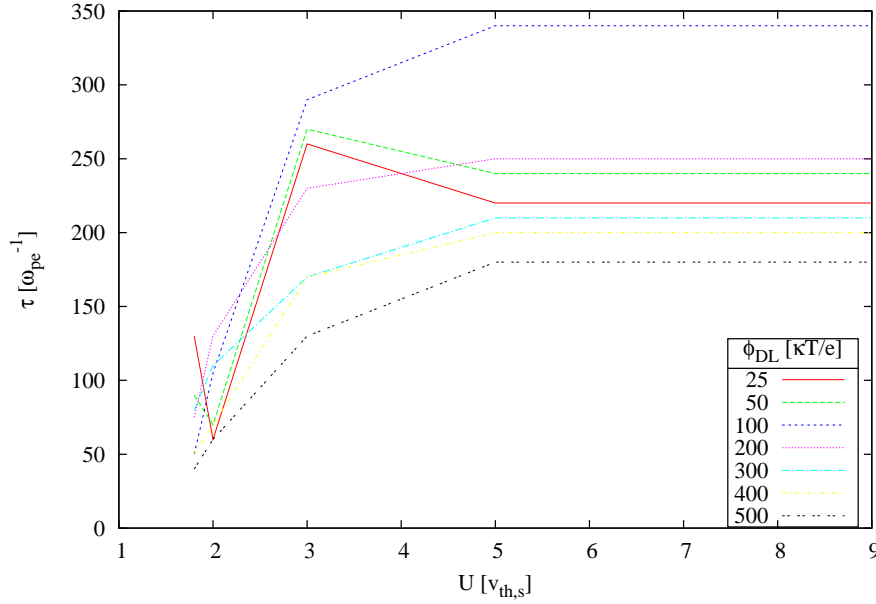


Figure 5.7: *Lifetime of simulations as functions of  $U$  for  $L = 350$  with one graph for each  $\phi_{DL} \in [25, 500]$ .*

Expanding lifetime from a function of  $L$  and  $\phi_{DL}$ , to a function of  $L$ ,  $\phi_{DL}$  and  $U$ ;  $\tau(L, \phi_{DL})$  to  $\tau(L, \phi_{DL}, U)$  allows for investigations of variations in  $\tau$  with  $U$  in addition to the variations already studied. Taking the lifetime  $\tau$  with variations in  $\phi_{DL}$  and  $U$ , for  $L = 350$  yields the results plotted in figure 5.7 where  $\tau$  is shown as functions of  $U$  for different  $\phi_{DL}$ . The lifetimes of each  $\phi_{DL}$  all have minimum values for  $U < 3$  and constant values for  $U \geq 5$ . For low  $\phi_{DL}$  ( $\phi_{DL} = 25$  and  $50$ ), the lifetime behaves differently, as these graphs have local maxima for  $U = 3$ .

### 5.2.3 Scaling law

The scaling law, equation (3.1), states that

$$\phi_{DL} < \eta L^2$$

must be true in order to have a stable double layer. The constant  $\eta$  is suggested to be of the order of 0.1 [Smith, 1982] and 0.04 [Singh, 1980]. Present simulations confirm the scaling law, and suggest that there should be a variation of  $\eta$  with the lifetime of the simulations.

Lifetimes of simulations (averaged over  $U$  and boundary conditions) for given sets of  $L$  and  $\phi_{DL}$  are presented in figure 5.6. This figure illustrates the general  $L$  and  $\phi_{DL}$  dependency as discussed in section 5.2.2. From the dataset illustrated by figure 5.6, points representing equi-lifetime contours  $\tau = 17, 25$  and  $35\omega_{pe}^{-1}$  are produced. These points are plotted in figure 5.8, with corresponding functional fittings in accordance with the scaling law. The fitting algorithm is the internal function `fit` in Gnuplot 4.2, which use the nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm.

The functions fitted,

$$\phi_{DL} = \eta_{\tau_0} L^2 \text{ for } \eta_{17} = 0.044, \quad \eta_{25} = 0.029 \text{ and } \eta_{35} = 0.013$$

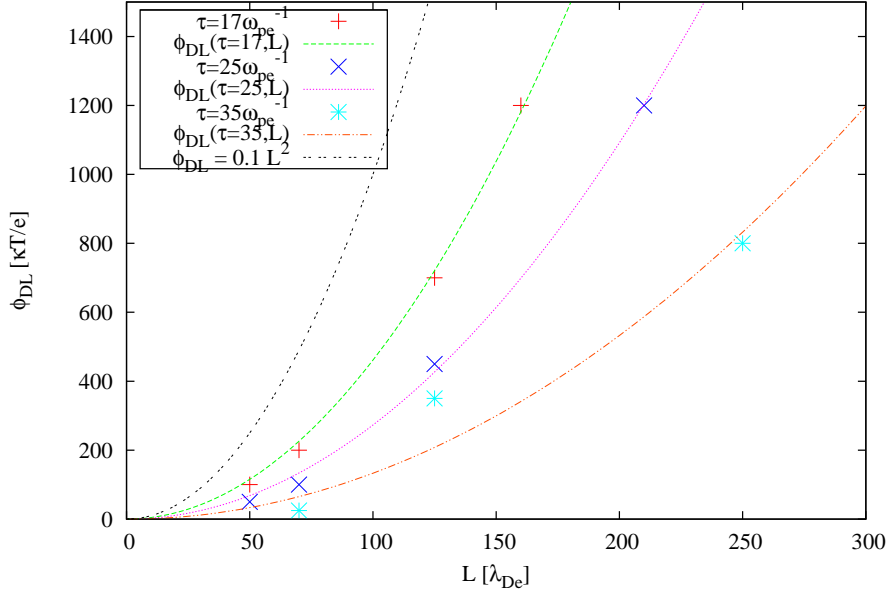


Figure 5.8: Combinations of  $L$  and  $\phi_{DL}$  where the lifetime  $\tau$  equals 17, 25 and  $35\omega_{pe}^{-1}$ . The functions  $\phi_{DL}(\tau = \tau_0, L) = \eta_{\tau_0} L^2$  have constants  $\eta_{17} = 0.044$ ,  $\eta_{25} = 0.029$  and  $\eta_{35} = 0.013$ .

agree with the points of figure 5.8. Note that the fitted parabolic functions are in better correspondence with the data points for decreasing  $\tau$ . Concluding from this fact and that the apparent shapes of the contours in figure 5.6, resembles parabolas only for  $\tau < 50$ , the scaling law can not represent equi-lifetime contours for  $\tau \gtrsim 50$ .

The scaling law suggested by [Smith, 1982], with  $\eta = 0.1$ , is plotted as a black dashed line in the same figure. As the lifetime shows a tendency to decrease towards the upper left corner of figure 5.8, it is evident that double layers are less likely to exist in this region of the  $L, \phi_{DL}$ -parameter space.

Judging from figure 5.6 and 5.8, the suggested values of  $\eta = 0.1$  and  $\eta = 0.04$  correspond to lifetimes of  $\tau \approx 10\omega_{pe}^{-1} = 1\omega_{pi}^{-1}$  and  $\tau \approx 20\omega_{pe}^{-1} = 2\omega_{pi}^{-1}$ , respectively.

## 5.3 Harmonic oscillations

Harmonic oscillations play an important role in double layer plasmas, either as a creation mechanism [Belova et al., 1980; Omura et al., 2008] or as waves growing in amplitude to the extent that simulations grow unstable. Either way, their characteristics are important for the existence of double layers.

In the following, temporal and spatial oscillations and their variations with different choices of boundary conditions and plasma parameters are investigated.

### 5.3.1 Temporal harmonic oscillations

For several simulations, the evolution of the finite potential leap,  $\phi_{DL}(t)$ , had clear similarities to harmonic oscillations. Examples of such can be seen in figure 5.1 for the D-vN and vN-vN boundary conditions.

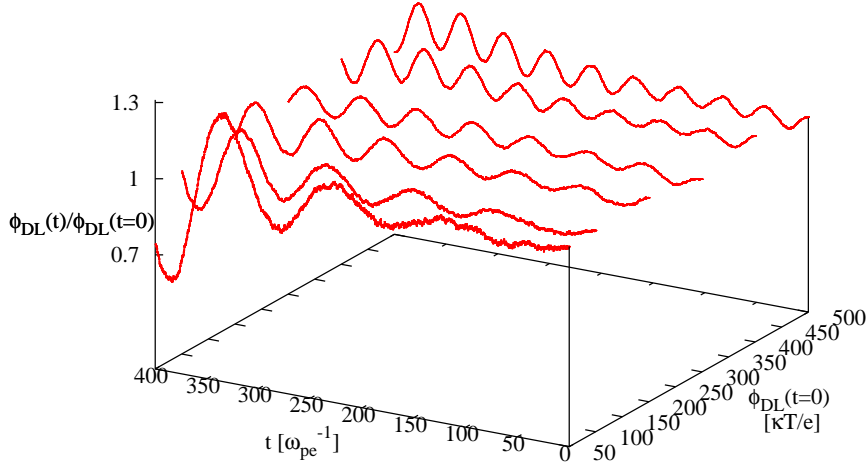


Figure 5.9: *The temporal evolution of the potential difference relative to the initialized potential difference,  $\phi_{DL}(\phi_{DL}(t=0), t)$ , for  $L = 500$ ,  $m_i/m_e = 100$ ,  $U = 5$  and  $vN$ - $vN$  boundary conditions.*

Both amplitude and frequency of such harmonic oscillations had variations with different simulation parameters. Figure 5.9 shows the relative temporal variation of the potential leap

$$\phi_{DL}(t)/\phi_{DL}(t=0) ,$$

plotted for  $L = 350$ ,  $U = 5$ ,  $vN$ - $vN$  boundary conditions and  $\phi_{DL}(t=0) \in [50, 500]$ .

The relative amplitude of perturbations,

$$\Delta\phi(t)/\phi_{DL}(t=0) \text{ where } \Delta\phi(t) = \phi_{DL}(t) - \phi_{DL}(t=0),$$

proves to be dependent on the initial  $\phi_{DL}$ ,  $\phi_{DL}(t=0)$ . Judging from figure 5.9 the relative amplitude is larger for lower  $\phi_{DL}(t=0)$  and decreasing for higher  $\phi_{DL}(t=0)$ .

The frequency of oscillations  $\omega$  can be written as a function of  $L$ ,  $\phi_{DL}$  and  $U$ ; i.e.  $\omega(L, \phi_{DL}, U)$ , measured in units of the electron plasma frequency,  $\omega_{pe}$ . The frequency is found to decrease with increasing  $L$ , increase with increasing  $\phi_{DL}$  and decrease with increasing  $U$ . The increase with  $\phi_{DL}$  can be directly observed from figure 5.10, where the total number of oscillations, hence also  $\omega$ , is observed to increase with increasing  $\phi_{DL}(t=0)$ . For  $\phi_{DL} \in [50, 500]$  and  $U \in [3, 9]$ , the observed values of  $\omega$  for  $L = 350, 500$  and  $750$  are

$$\omega(L = 350, \phi_{DL}, U) \in [0.1, 0.22] \quad \bar{\omega}(L = 350) = 0.16 , \quad (5.4)$$

$$\omega(L = 500, \phi_{DL}, U) \in [0.09, 0.18] \quad \bar{\omega}(L = 500) = 0.135 , \quad (5.5)$$

$$\omega(L = 750, \phi_{DL}, U) \in [0.06, 0.13] \quad \bar{\omega}(L = 750) = 0.095 , \quad (5.6)$$

Double layers with width  $L \leq 250$  has been left out of this discussion as harmonic oscillations are either rare or short lived.

The  $\phi_{DL}$  and  $U$  variations are demonstrated in figure 5.10 for  $L = 350$ . Plots for  $L = 500$  and  $750$  (not shown here) have the same shape but with different average values

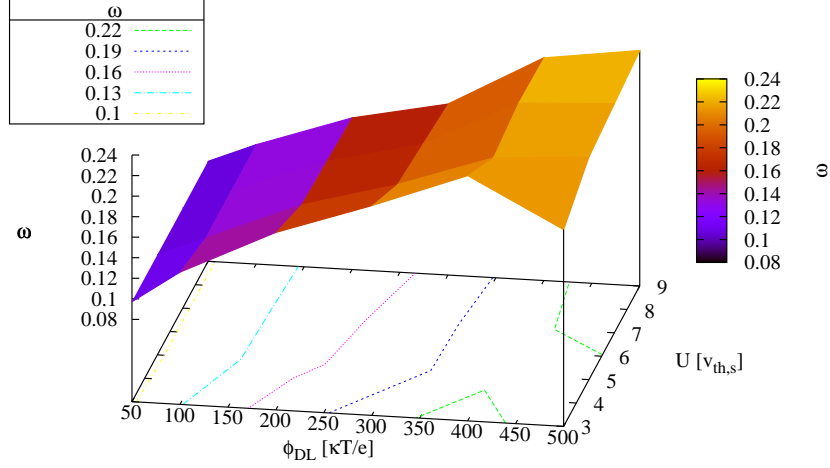


Figure 5.10: The angular frequency  $\omega(L = 350, \phi_{DL}, U)$  with contours of  $\omega$  at the base of the figure.

as indicated by equations (5.4)-(5.6). All observed angular frequencies are of the order 0.1 which can be recognized as the ion plasma frequency,  $\omega_{pi} = 0.1\omega_{pe}$ .

For most simulations, oscillations observed in  $\phi_{DL}(t)$  have an exponential growth in amplitude. Such oscillations are considered to be unstable as there is no apparent upper limit for the oscillations. Examples of exponentially increasing amplitudes can be seen in the  $\phi_{DL}(t)$  plots of figure 5.1, for the D-vN and vN-vN boundary conditions.

In some simulations, however, the observed waves are damped. Damped waves indicate that the plasma conditions represented by such simulations are favourable for the existence of double layers. Figure 5.5 gives an example of damped waves in the  $\phi_{DL}(t)$  plots for the vN-D boundary condition case. In present simulations, damped oscillations are only observed for vN-D boundary conditions and  $L > 250$ .

### 5.3.2 Spatial harmonic oscillations

Spatial oscillations are observed to appear in the initially homogeneous plasmas added to each side of the double layer. Figure 5.11 show the power of these oscillations, traced as functions of time, for the simulation with  $L = 500$ ,  $\phi_{DL} = 500$ ,  $m_i/m_e = 100$ ,  $U = 3$  and D-D boundary conditions. The power spectra were calculated with a Fast Fourier Transform (FFT). Details of the FFT procedure and how figure 5.11 was created is given in section 4.8.5.

Studying power spectra from simulations of  $L \in [250, 750]$ ,  $\phi_{DL} \in [100, 500]$ ,  $m_i/m_e = 100$  and  $U \in [3, 7]$  proved that the observed values of the wave numbers had no variations with  $\phi_{DL}$  or  $U$ . For variations in  $L$ , on the other hand, it was found that the value of wave numbers decreased with increasing  $L$ .

Table 5.2 shows the observed values of  $k$  and corresponding wavelengths  $\lambda$  for  $L \in [250, 750]$ . These quantities are also compared to the lowest detectable value of  $k$  (due to the finite size of the analysed region),  $k_{min}$ , and the system length,  $L_{sys} = L + 500\lambda_{De}$ , respectively.

The graphical representation of  $k_{min}$ ,  $k_1$  and  $k_2$  in figure 5.12 shows the decrease of

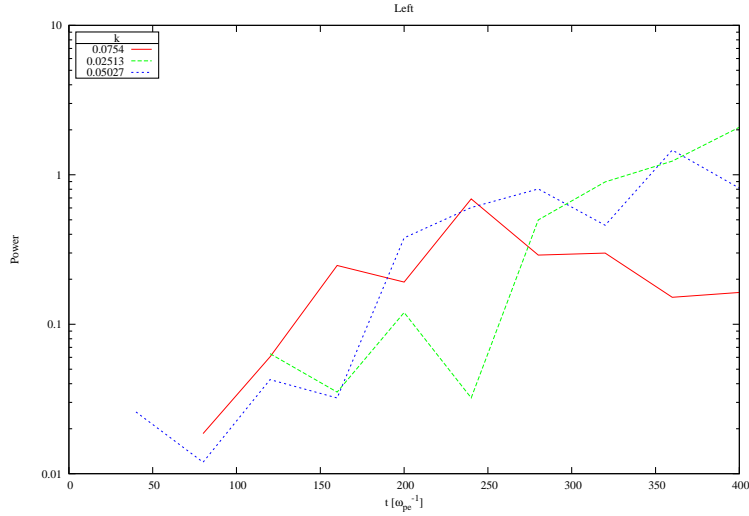


Figure 5.11: Tracing the power of dominating peaks in the power spectrum as functions of time at the left hand side of the double layer. Simulation parameters are  $L = 500$ ,  $\phi_{DL} = 500$ ,  $m_i/m_e = 100$ ,  $U = 3$  and D-D boundary conditions.

	$L = 250, k_{min} = 0.03351$				$L = 350, k_{min} = 0.02957$			
	$k$	$k/k_{min}$	$\lambda$	$\lambda/L_{sys}$	$k$	$k/k_{min}$	$\lambda$	$\lambda/L_{sys}$
$k_1$	0.03351	1	188	0.250	0.02957	1	212	0.250
$k_2$	0.06702	2	94	0.125	0.05910	2	106	0.125

	$L = 500, k_{min} = 0.02513$				$L = 750, k_{min} = 0.04021$			
	$k$	$k/k_{min}$	$\lambda$	$\lambda/L_{sys}$	$k$	$k/k_{min}$	$\lambda$	$\lambda/L_{sys}$
$k_1$	0.02513	1	250	0.250				
$k_2$	0.05027	2	125	0.125	0.04021	2	156	0.125

Table 5.2: Observed spatial oscillations for simulations with  $L \in [250, 750]$ . The lowest detectable  $k$  by the Fourier analysis,  $k_{min}$ , is given for each  $L$ . Oscillation modes  $k_1$  and  $k_2$  are listed with their respective wavenumbers  $k$ , measured in  $\lambda_{De}^{-1}$  and  $k_{min}$ , and wavelengths  $\lambda$  measured in  $\lambda_{De}$  system length  $L_{sys} = L + 500\lambda_{De}$ .

wave numbers with increasing system length. Two linear functions,  $k_1(L)$  and  $k_2(L)$ , are fitted to the measured values of  $k_1$  and  $k_2$ , proving the linear decrease of  $k_1$  and  $k_2$  with  $L$ . An estimate has been made of the  $k_1$  wave mode for  $L = 750$ , where it has been assumed half the value of  $k_2$ . This estimate is given as a blue diamond in figure 5.12. Since the minimum detectable wave number is higher than this estimate, it could not be observed by the present Fourier analysis.

The observed values of  $k_1$  and  $k_2$  are observed to follow the linear trend of  $k_{min}$  for  $L < 750$ . It is possible that the observed linear decrease of  $k_1$  and  $k_2$  are products of the linear decrease of  $k_{min}$ . The decrease of  $k_{min}$  is a due to a linear increase in  $L_{FFT}$  caused by the present implementation of the FFT routine. On the other hand, these results do not exclude that there might be a variation in  $k$  due to the change in  $L$ . To investigate this possibility is a natural and interesting choice of future work based on this thesis.

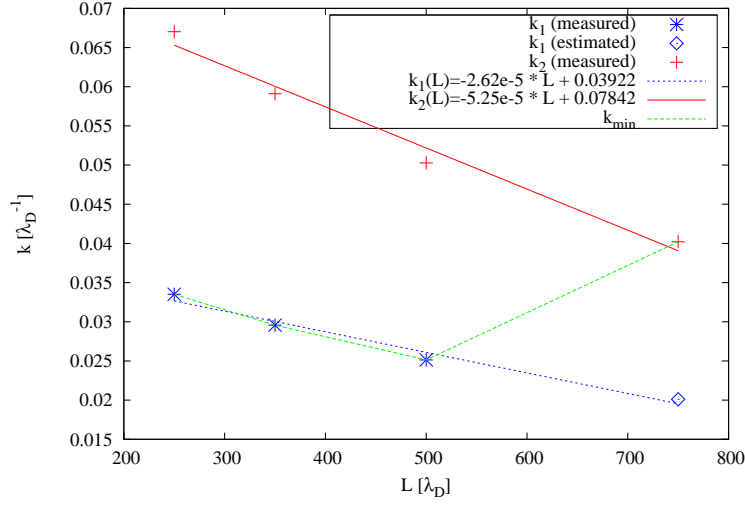


Figure 5.12: Measured and estimated values of  $k_1$ , measured values of  $k_2$ , linear fits of  $k_1(L)$  and  $k_2(L)$  to  $k_1$  and  $k_2$  data points, and the lowest measurable  $k$  with the present implementation of FFT,  $k_{min}$ .

Waves with  $k$  such that

$$\frac{kU}{\omega_{pe}} < \frac{k_c U}{\omega_{pe}} = K_c = \sqrt{1 + 3(m_e/m_i)^{1/3} + 3(m_e/m_i)^{2/3} + m_e/m_i} = 1.17$$

are said to be subject of the Buneman instability in the representative simplified model discussed in section 3.3.4. Assuming the highest value for  $U$  used in the present simulations,  $U = 9$ , and inserting this into the equation above, all waves with  $k < k_c = 0.13$  are expected to be unstable. Waves with  $k > k_c$  were very seldom observed in the present results.

## 5.4 Phase space vortices

In some of the present simulations, instabilities here interpreted as the Buneman instability, saturate into a series of ion and/or electron holes [Schamel, 1986] in the initially homogeneous plasmas added to each side of the double layer. If allowed to grow, these holes will dominate the particle interactions through trapping of particles with less energy than the potential well energy  $e\Delta\phi$  of the hole.

An electron/ion hole seen in an electron/ion phase space diagram resembles a vortex, spiraling in towards some coordinate  $(x, v)$  if  $x$  and  $v$  is the position and velocity of the hole. In an ion/electron phase space diagram, an electron/ion<sup>1</sup> hole is recognized as a void at the coordinate  $(x, v)$ . Observations of phase space vortices have been made in both experiments [Saeki et al., 1979; Pécseli et al., 1984] and in numerical simulations [Berk et al., 1970; Lynov et al., 1979; Pécseli et al., 1984].

Examples of such vortices are shown in figure 5.14 for the simulation  $L = 125$ ,  $\phi_{DL} = 400$ ,  $m_i/m_e = 100$ ,  $U = 5$  and D-D boundary conditions. This figure illustrates the ion and electron holes in several ways. In the top panel (stacked field plots) the hole shapes are directly observed in the potential profiles, with corresponding charge density

<sup>1</sup>Note the reversed sequence electron/ion

and electric field. In the lower panel, the mentioned vortex shapes are fully developed for electrons and only just recently appeared for ions at  $t = 100\omega_{pe}^{-1}$ . Figure 5.13 shows the power of the spatial oscillations, here interpreted as series of ion and electron holes. The power is seen to increase continuously with time for all the wave modes observed.

Typically, these structures occur for double layers with  $\phi_{DL} \gtrsim 0.03L^2$  which can be recognized as those double layers which according to the scaling law are unstable. There were no visible trends for which values of  $U$  encourages phase space vortices, i.e. all  $U \in [1.8, 9]$  appear equally likely to induce Buneman instabilities. With boundary conditions the system response was slightly different, as the two combinations D-D and vN-D repeatedly were observed with phase space vortices, while the others were not.

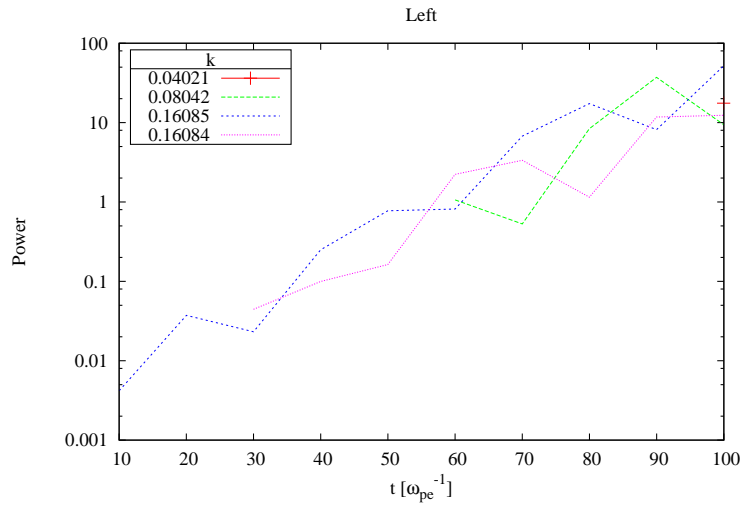


Figure 5.13: *The temporal evolutions of the power of spatial oscillations, here interpreted as series of ion and electron holes, for the simulation  $L = 125$ ,  $\phi_{DL} = 400$ ,  $m_i/m_e = 100$ ,  $U = 5$  and D-D boundary conditions.*

All the observed cases of phase space vortices were seen at the left hand side of the double layer, where accelerated particles have the lowest drift velocities. This observation is in contradiction with previous studies, where these phase space structures are expected to appear at the opposite side where electrons have higher velocities. Also, these structures can form long lived non-linear plasma equilibria as described by the general BGK-formalism. The vortex-like phenomena will not be discussed any further here, but in a general analysis they belong naturally to the class of stationary BGK-solutions.



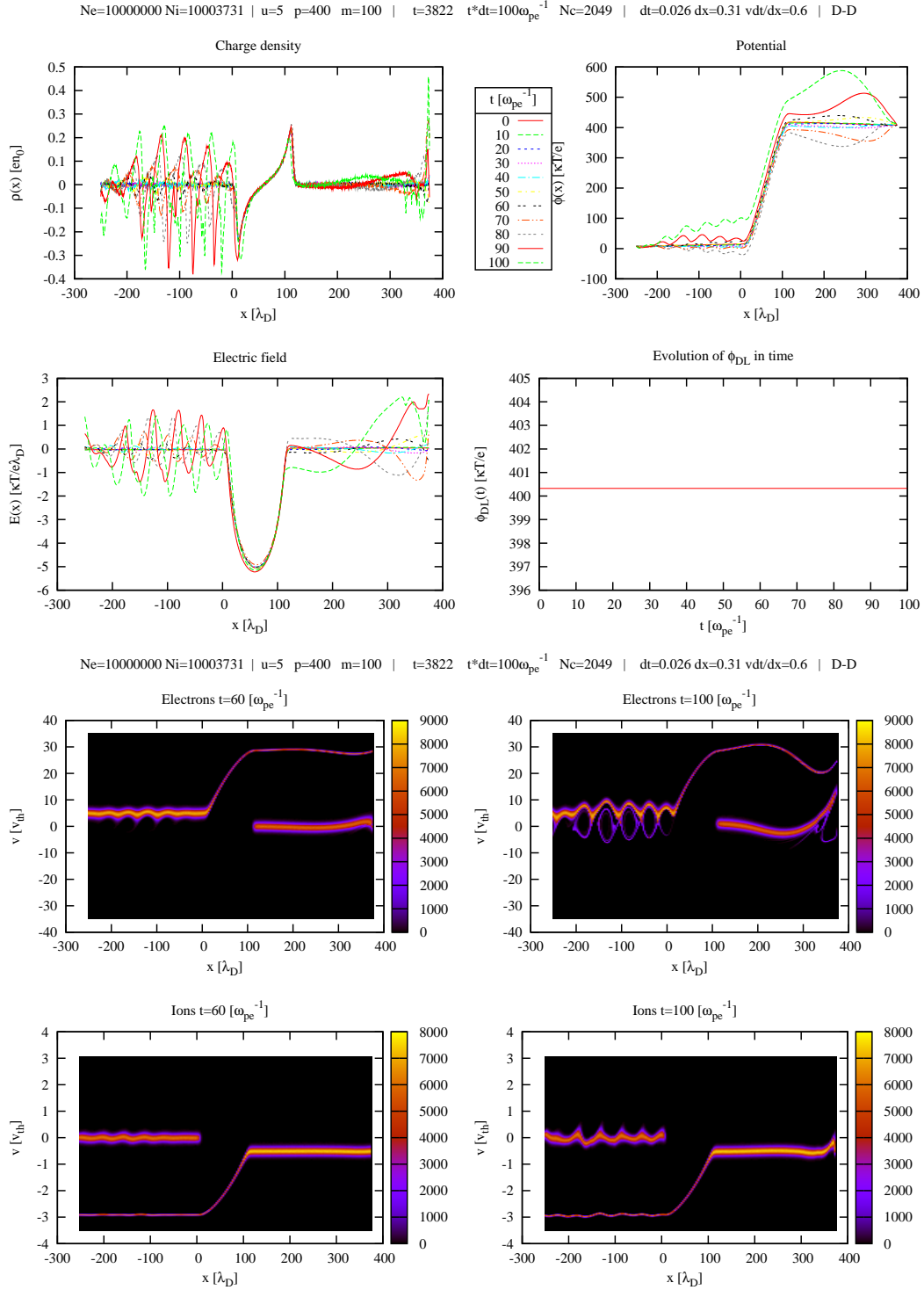


Figure 5.14: Field stack plots and phase space diagrams for  $L = 125$ ,  $\phi_{DL} = 400$ ,  $m_i/m_e = 100$ ,  $U = 5$  and D-D boundary conditions.

[illegible]

Table 5.3: The combinations of  $L$ ,  $\phi_{DL}$ ,  $m_i/m_e$  and  $U$  which has been simulated are marked with a  $\bullet$  or a number referring to a figure in the text. The reference simulation is marked with red font.

## Chapter 6

# Discussions and Conclusions

In this chapter, the results presented in chapter 5 are discussed in the context of the plasma conditions supporting the existence of a double layer. The discussion is summarized before the possibilities for the future work is listed. Notations and parameters used in this chapter are as defined in chapter 5.

### 6.1 Boundary conditions

Three forms of boundary conditions have been discussed in the earlier chapters: The initial probability density functions at the boundaries, the in- and outflux of particles and the boundary conditions on the potential. As none of the results suggest that the choice of boundary probability density functions (centered and shifted Maxwellians) or particle flux are unphysical or particularly significant, these topics will not be discussed any further. The boundary conditions on the potential, however, proved to have a noticeable effect on the results. The four effects of the boundary condition combinations can be summarized as follows:

**D-D** Remarkably stable simulations for  $\phi_{DL} < 200$ . For  $\phi_{DL} \geq 200$ , perturbations were induced, with increasing amplitudes for increasing  $\phi_{DL}$ . For  $\phi_{DL} \geq 500$ , D-D appeared highly unstable.

**D-vN** Equal trend as for D-D with increasing  $\phi_{DL}$ . However, D-vN always induced higher amplitude perturbations than D-D.

**vN-D** Overall the most stable boundary condition combination. Induced less perturbations for  $\phi_{DL} \geq 350$  than for  $\phi_{DL} < 350$ .

**vN-vN** Remarkably stable simulations for  $\phi_{DL} \geq 500$ . The relative amplitudes of the perturbations were increasing with decreasing  $\phi_{DL}$ .

Perturbations were more frequently induced by Dirichlet type boundaries than those of von Neumann. These wave-like perturbations appeared close to the boundary, clearly separated from the double layer. Studying the temporal evolution of such waves show that they propagate inwards from the boundary to the double layer. An example of such waves is shown in figure 5.2b, propagating rightwards from the left hand side, Dirichlet type, boundary. Some of these waves had the properties of standing waves, as seen in the charge density stack plots of the D-D boundary condition panels of figure 5.5. This particular example shows a standing wave in the left hand side plasma of wavelength equal to the added plasma's length. No standing waves as shown here were observed in connection to von Neumann boundaries.

From these observations it is possible to conclude that Dirichlet (contrary to von Neumann) type boundaries encourage perturbations to grow in amplitude and thus form waves as seen in the present results. A possible explanation is as follows: Perturbations located just inside of a Dirichlet boundary will create a strong gradient in the potential between the fixed boundary potential value and the perturbation. The gradient, physically equal to the presence of an electric field, will then accelerate particles, further enhancing the effect of the perturbation. In the opposite case, with von Neumann boundaries, the boundary value of the potential will adjust itself so that the electric field at the boundary equals zero. Thus further enhancements of the perturbation are not encouraged by the boundary condition.

The combination vN-vN represents a very interesting scenario as the boundaries are set free to adjust themselves, requiring only zero electric field. The boundary conditions vN-vN allows the system to make small adjustments to the potential profile without introducing electric fields at the boundary. Assuming a more stable configuration of the double layer exists, close to but not equal the initialized configuration, the system is able to adjust itself towards the more stable alternative.

The boundary condition combination vN-D represents an interesting class of double layers of two reasons; 1) Double layers are overall highly stable for the vN-D combination, and 2) they inhibit damping of the temporal oscillations which is not seen for any other combination. The fact that oscillations are seen to be damped contributes to the conclusion that this particular combination represents a highly stable class of double layers. None of the experimental double layer studies found have been able to confirm or deny the presence of vN-D boundary conditions in nature.

As mentioned earlier in section 2.4.4, the observed properties for D-D and vN-vN boundary conditions can be assigned to plasmas of either fixed potential or constant current at the boundaries, respectively. The results found in the present study proved that D-D was the most stable combination for  $\phi_{DL} \lesssim 200$ , vN-vN the most stable for the opposite. It is therefore possible to conclude from these observations that for  $\phi_{DL} \lesssim 200$ , the double layer is more stable if in contact with plasmas on both sides of fixed potentials. Likewise, for  $\phi_{DL} \gtrsim 200$ , the double layer is more stable if the plasma have e.g. high conductance, so that the current is kept constant.

Which physical phenomena the remaining combinations, D-vN and vN-D, represent, has not been found in the present studies. Therefore it is not possible at this point to assign the observed properties to a known physical phenomena. Still, the properties are presented here may in the future be used if observations of double layers can be seen to match the boundary condition combinations that were investigated.

## 6.2 Plasma parameters

Simulations represent different plasma conditions through the variations of the parameters, listed in table 5.1. The lifetime of the simulation, as defined in section 5.2, can directly be interpreted as whether such a double layer can exist or not. The maximum simulation time was  $t = 1000\omega_{pe}^{-1}$ .

### Double layer width

Lifetime was found to increase with increasing  $L$ . For the narrowest double layers simulated,  $L = 50$  and  $70$ , lifetimes was typically in the order of  $10\omega_{pe}^{-1} = 1\omega_{pi}^{-1}$ , increasing to equal the maximum simulation time,  $\tau = 1000\omega_{pe}^{-1}$ , for  $L = 750$ . It is likely that these simulations are able to last even longer. The increase of the lifetime with  $L$  was found

to be linear (equation 5.1), i.e.

$$\tau(L, \phi_{DL}) \sim L \quad \text{for } \phi_{DL} \in [50, 800] .$$

In the previous simulations with initially homogeneous plasmas, the double layers studied were created from two-stream (i.e. Buneman) instabilities [Goertz and Joyce, 1975; Belova et al., 1980]. The double layers created had a typical width in the order of  $10 - 100\lambda_{De}$ , which is smaller by a factor of 10 compared to the double layers found to be stable in the present simulations. The double layers of the previous simulations were found to highly transient in nature. Results from present simulations, however, proved that double layers longer than those simulated by [Goertz and Joyce, 1975; Belova et al., 1980] were necessary to have time stationary simulations. It is therefore possible that a study similar to [Goertz and Joyce, 1975; Belova et al., 1980] will be able to create time stationary double layers from a Buneman unstable, homogeneous plasma, if the system lengths are comparable to that of the present simulations.

### Double layer strength

It was found that the lifetime of the simulations had weak variations with  $\phi_{DL}$ , compared to what was observed with  $L$ . Yet however small, the variations were present and the observed variation of the lifetime  $\tau(L, \phi_{DL})$  with  $\phi_{DL}$  are given by (equations (5.2) and (5.3))

$$\begin{aligned} \tau(L, \phi_{DL}) &\sim -\phi_{DL}^2 \quad \text{for } L \in [50, 250] , \\ \tau(L, \phi_{DL}) &\sim \phi_{DL} \quad \text{for } L \in [350, 750] . \end{aligned}$$

As all double layers simulated are of the type *strong* double layers, the results from each simulation are expected to share the main characteristics typical of strong double layers. It is therefore natural that there is little variation in the results as long as  $e\phi_{DL} \gg \kappa T$ .

In the previous numerical studies mentioned in section 3.6, the double layers were in general weaker than those simulated here. However, the double layer width was also increased compared to the previous so that the ratio  $\phi_{DL}/L^2$  was approximately constant, and therefore can be assumed to represent similar classes of double layers.

### Mass ratio

Response time of the plasma is expected to be governed by the long ion response time, due to their high inertia. Seen from this perspective, the variation of lifetime with mass ratio is expected to follow the relationship  $\tau \sim \omega_{pe}/\omega_{pi} = \sqrt{m_i/m_e}$ .

Simulations with a mass ratio of unity had, as expected, a very short lifetime. The ions no longer had higher inertia than electrons, thus the response times were short. Unity mass ratio may represent more exotic conditions like an electron-positron plasma. However, such examples are not of interest in the present studies. Simulations with unity mass ratio served as a consistency test of the code, as the only asymmetry of the simulation was due to possible asymmetries imposed by the boundary conditions.

Increasing the mass ratio from unity to a mass ratio of  $m_i/m_e = M > 1$ , had the expected effect of the increased response time of the plasma. The lifetime of otherwise two identical simulations, with mass ratios of 1 and  $M$  had the relationship  $\tau_M = \sqrt{M}\tau_1$ . Two values of  $M > 1$  were tested in the present simulations,  $M = 100$  and  $M = 300$ . The results of which showed no other difference than those expected.

### Drift velocity

The Bohm criterion sets an absolute lower value of the drift velocities of accelerated particle populations of a double layer. The Bohm criterion is summarized by [Smith,

1982], given by equations (3.3) and (3.4). As particles move adiabatically in double layers [Raadu, 1989] and the temperature ratio of present simulations are set to unity, the proportionality factors  $\gamma_s + T_r/T_s = 3 + 1 = 4$  for both species. Thus comparing the Bohm criterion,  $U_s \geq 4v_{ths}$ , with the results given in figure 5.7, it is obvious that these correspond to each other. The present results also introduce additional effects when the simulated velocities are close to the Bohm velocity: The lifetimes of  $\phi_{DL} = 25$  and 50 are observed to have a maximum value for  $U = 3$ . A possible explanation for this anomaly is that the low  $\phi_{DL}$  double layers may have inherited some characteristics of weak double layers, where low drift velocities are more common than with strong double layers. This is, however, in contradiction with the conclusion based on the small variations with  $\phi_{DL}$ , that all simulated double layers share typical strong double layer characteristics.

### 6.3 Scaling law

The most significant individual result of the present simulations is the confirmation of the scaling law suggested in previous simulations [Smith, 1982]. The scaling law predicts that double layers with  $\phi_{DL} < \eta L^2$  forms a stable branch of the  $L, \phi_{DL}$ -parameter space. This is qualitatively confirmed from the present results found in section 5.2.3. From figures 5.6 and 5.8, an alternative interpretation of the scaling law can be made:

Figure 5.8 demonstrates that for suitable choices of  $\eta$ , the scaling law can represent equi-lifetime contours of figure 5.6. As the relative error of the functional fittings increase with increasing lifetime, it is likely that the scaling law only can represent contours for lifetimes  $\tau \lesssim 35\omega_{pe}^{-1}$ .

The scaling law as predicted by [Smith, 1982], i.e.  $\eta = 0.1$ , corresponds to a lifetime of  $\tau = 10\omega_{pe}^{-1}$  in the present simulations. To state that the suggested scaling law forms an exact limit between stable and unstable simulations, corresponds to saying that simulations with  $\tau \leq 10\omega_{pe}^{-1} = 1\omega_{pi}^{-1}$  are *unstable*. In general, limits defining what is stable or not, will not be as absolute or discrete as the scaling law appears. However, the statement given above, that a plasma configuration is unstable if the ions are unable to complete more than one oscillation, appears logical and intuitive.

### 6.4 Buneman instability

The simplified model of the Buneman instability, given in section 3.3.4, states that waves with wave numbers lower than a critical value  $k_c$  will be subject to the Buneman instability. Almost all of the observed waves in the present results have wave numbers  $k < k_c$  and are therefore expected to behave as predicted by the Buneman instability theory.

The wave-like perturbations, recognized as Buneman instability waves, appear in all cases at the low potential side of the double layer. This observation is in contradiction to the results of previous simulations [Smith, 1982; Ergun et al., 2003]<sup>1</sup>. These studies argue for the presence of two-stream (i.e. Buneman) instabilities at the high potential side of the double layer, where electrons have higher velocities as they have been accelerated through the double layer. It is not known how the boundary conditions of both previous and present simulations affect the results, which may be one reason for why the results of previous and present studies show to different results.

In section 5.4, the Buneman instability was shown to saturate into ion- and electron holes, recognized by their phase space vortices. These phase space vortices were observed only for double layers with  $\phi_{DL} \gtrsim 0.03L^2$ , a relationship which is recognized as the opposite of the scaling law, section 5.2.3. One can thus conclude that double layers with

<sup>1</sup>See figure 7 of [Ergun et al., 2003].

$\phi_{DL} \gtrsim 0.03L^2$  will experience large amplitude perturbations, created by to the Buneman instability, compared to double layers with  $\phi_{DL} \lesssim 0.03L^2$ . These amplitudes grow large enough for the simulation to be regarded unstable (by the definition in section 5.2) within a short period of time.

Harmonic generation and wave mixing is seen to occur for many of the Buneman instability waves. The power of the initial wave increases until more waves appear. Energy is then transferred from the initial to the new waves, and the power of the initial wave decrease while increase for the new waves. The set of waves observed for all present simulations are proportional to each other by integer constants;  $k_2 = 2k_1$ ,  $k_3 = 3k_1$ ,  $k_3 = k_2 + k_1$ , etc. Which of the wave modes  $k_1$ ,  $k_2$ ,  $k_3$ , ... is the initial wave appears to be random.

The values of  $k_1$  and  $k_2$  were observed to decrease with increasing  $L$ . It was also observed that the decrease of these wave numbers followed exactly the decrease in the minimum detectable wave number  $k_{min}$ . As mentioned in sections 4.8.5 and 5.3.2, the value of  $k_{min}$  is a product of the present implementation of the Fourier analysis. It is therefore a possibility that the observed decrease in  $k_1$  and  $k_2$  is influenced by the model used, and not (just) a plasma physical result. An alternative approach if a possibility to repeat such analysis occurs would be to choose the more general Fourier analysis and not the Fast Fourier Transform. There are two advantages to this approach compared to the present Fourier analysis. First, the width of the studied region,  $L_{FFT}$ , can be kept constant as the size of the analyzed array no longer must be of the form  $N = 2^n$  for  $n = 1, 2, 3, \dots$ . Second, the wave numbers identified will not be discrete values  $k = 2\pi i/L_{FFT}$  for  $i = 0, 1, 2, 3, \dots$ . A clear disadvantage of the general Fourier transform is the increased computational time compared to the Fast Fourier Transform.

Assuming that the temporal and spatial oscillations, presented in sections 5.3.1 and 5.3.2, represent the same wave propagating through the double layer, it is possible to estimate the phase velocity of those waves. Taking the typical value of the two lowest wave numbers,  $k \approx 0.03$  and  $0.06\lambda_{De}^{-1}$  and likewise for frequency,  $\omega \approx 0.1\omega_{pe}$  gives the phase velocities  $\omega/k = 0.1\omega_{pe}/0.03\lambda_{De}^{-1} = 3.3v_{the}$  and  $\omega/k = 0.1\omega_{pe}/0.06\lambda_{De}^{-1} = 1.7v_{the}$ .

## 6.5 Summary

Variations in boundary conditions performed in the present studies, proved the influence boundary conditions can have on the results. Dirichlet (D) boundary conditions was in general seen to induce larger amplitude perturbations than von Neumann (vN). The combination D-D proved to be more stable for double layer strengths of the order  $\phi_{DL} < 200\kappa T/e$ , while vN-vN proved to be more stable for  $\phi_{DL} > 200\kappa T/e$ . These combinations represent, as mentioned in section 2.4.4, constant voltage and constant current generators in a classical lumped electronic circuit. It is therefore likely that double layers in nature, where the surrounding plasma can be approximated by the mentioned generators, have corresponding boundary conditions according to the above given ranges of  $\phi_{DL}$ . The earlier mentioned examples of such plasmas are either to adjacent plasmas of fixed potential of plasmas of high conductivity, forcing a constant current. It was also observed that the combination vN-D inhabited damped oscillations, indicating that the given boundary conditions represent a highly stable class of double layers.

The variations in the plasma parameters (double layer width,  $L$ , strength of potential leap,  $\phi_{DL}$ , particle mass ratio,  $m_i/m_e$  and drift velocities of accelerated particles,  $U$ ) proved, most importantly, two well established theories in double layer physics; the scaling law and the Bohm criterion. It was also found that wide double layers, i.e.  $L > 300\lambda_{De}$ , form a remarkably stable branch of double layers.

The Buneman instability was frequently observed in the plasmas added to each side

of the double layer. The instabilities had wave-like properties, and their wave numbers were found to decrease with increasing double layer width,  $L$ . The present studies were not able to say if this decrease was caused by the method used or a plasma physical phenomena. A saturation of these instabilities was observed for double layers predicted to be unstable by the scaling law. These saturations formed ion- and electron holes recognized as vortices in the phase space diagrams.

The present studies have given an overview the plasma conditions, represented by the choice of plasma parameters or boundary conditions, for which double layers are able to exist. The study of four different combinations of boundary conditions, compared to most previous studies which investigated only one combination, gave an understanding of which of the effects were produced by boundary conditions, and not by physical properties of the plasma. The reliability of the numerical model further was strengthened by the confirmations of the Bohm criterion and the scaling law. The present study contributes to a better understanding of the scaling law, as existing literature on the topic have many unclear details with regards to this theory. Finally, the present study includes results where the Buneman instability appear on the opposite side of the double layer compared to previous studies, thus allowing for alternative models of the Buneman instability's role in double layers.

## 6.6 Future work

As time is limited for a masters degree, many projects and goals had to be left behind in order to finish the work in due time. If, however, the opportunity bids itself to continue on this project, the following improvements would be included into the numerical simulations:

- Explore possibilities to why certain boundary conditions resulted in more stable simulations.
- Expand the numerical model to include weak double layers, which can be interesting also for laboratory experiments.
- Investigate the results observed for the spatial oscillations with regular Fourier transforms instead of Fast Fourier Transforms, aiming to exclude the method as a possible cause for the observed effects.
- Increase the number of spatial dimensions.
- Introduce effects of magnetic fields and electromagnetic effects.
- Explore plasmas of non unity temperature ratios.
- Explore double layers where the Langmuir condition is violated, double layers which are said to cause radio frequency waves [Hubbard and Joyce, 1979].
- Compare the results of this theoretical model to recent observations of astrophysical and space plasmas.



# Bibliography

- Alfvén, H. (1958). On the theory of magnetic storms and aurorae. *Tellus*, 10:104.
- André, M., editor (1993). *The Freja Scientific Satellite*. Swedish Institute of Space Physics, Kiruna, Sweden, 3. edition. IRF Scientific Report 214.
- Belova, N. G., Galeev, A. A., Sagdeev, R. Z., and Sigov, I. S. (1980). Collapse of the electric field in double layers. *ZhETF Pis ma Redaktsiiu*, 31:551–555.
- Berk, H. L., Nielsen, C. E., and Roberts, K. V. (1970). Phase space hydrodynamics of equivalent nonlinear systems: experimental and computational observations. *Phys. Fluids*, 13:980–995.
- Bernstein, I. B., Greene, J. M., and Kruskal, M. D. (1957). Exact nonlinear plasma oscillations. *Physical Review*, 108(3):546–550.
- Birdsall, C. K. and Langdon, A. B. (1985). *Plasma Physics Via Computer*. McGraw-Hill, Inc., New York, NY, USA.
- Bishop, A. S. (1958). *Project Sherwood The U.S. Program in Controlled Fusion*. Addison-Wesley Publishing Company, inc., Reading, Massachusetts, USA.
- Block, L. P. (1972). Potential double layers in the ionosphere. *Cosmic Electrodyn., Vol. 3*, p. 349 - 376, 3:349–376.
- Block, L. P. (1978). A double layer review. *Astrophysics and Space Science*, 55:59–83.
- Bohm, D. (1949). *The Characteristics of Electrical Discharges in Magnetic Fields*, chapter Minimum Ionic Kinetic Energy for a Stable Sheath, page 77. MCGraw-Hill, Inc., New York, USA.
- Box, G. and Muller, M. (1958). A note on the generation of random number deviates. *Annals Math. Stat.*, 29:610–611.
- Chen, F. F. (1984). *Introduction to Plasma Physics and Controlled Fusion*, volume 1. Plenum Press, New York and London.
- Courant, R., Friedrichs, K., and Lewy, H. (1928). On the partial difference equations of mathematical physics. *Mathematische Annale*, 100:32–74.
- Davidson, R. C. (1972). *Methods in Nonlinear Plasma Theory*, volume 37 of *Pure and Applied Physics*. Academic Press, New York and London.
- Ergun, R. E., Andersson, L., Carlson, C. W., Newman, D. L., and Goldman, M. V. (2003). Double layers in the downward current region of the aurora. *Nonlin. Processes Geophys.*, 10:45–52.

- Goertz, C. K. and Joyce, G. (1975). Numerical Simulation of the Plasma Double Layer. *Astrophysics and Space Science*, 32:165–173.
- Horowitz, P. and Hill, W. (1980). *The Art of Electronics*. Cambridge University Press, Cambridge, New York, USA.
- Hubbard, R. F. and Joyce, G. (1979). Simulation of auroral double layers. *Journal of Geophysical Research*, 84:4297–4304.
- Johnson, L. E. (1980). Numerical model of plasma double layers using the vlasov equation. *Journal of Plasma Physics*, 23:433–452.
- Jovanović, D., Lynov, J. P., Michelsen, P., Pécseli, H. L., Rasmussen, J. J., and Thomsen, K. (1982). Three dimensional double layers in magnetized plasmas. *Geophys. Res. Lett.*, 9:1049–1052.
- Joyce, G. and Hubbard, R. F. (1978). Numerical simulation of plasma double layers. *Journal of Plasma Physics*, 20:391–404.
- Klimontovich (1967). *The statistical Theory of Non-Equilibrium Processes in a Plasma*. The Massachusetts Institute of Technology Press, Cambridge, Massachusetts, USA.
- Langmuir, I. (1929). The Interaction of Electron and Positive Ion Space Charges in Cathode Sheaths. *Physical Review*, 33:954–989.
- Ledvina, S. A., Ma, Y.-J., and Kallio, E. (2008). Modelling and simulating flowing plasmas and related phenomena. *Space Sci. Rev.*, 139:143–189.
- Lynov, J. P., Michelsen, P., Pécseli, H. L., Rasmussen, J. J., Saeki, K., and Turikov, V. (1979). Observations of solitary structures in a magnetized, plasma loaded waveguide. *Phys. Scripta*, 20:328–335.
- McIlwain, C. E. (1960). *Direct Measurement of Particles Producing Visible Aurorae*. PhD thesis, The University of Iowa.
- Mozer, F. S., Carlson, C. W., Hudson, M. K., Torbert, R. B., Parady, B., Yatteau, J., and Kelley, M. C. (1977). Observations of paired electrostatic shocks in the polar magnetosphere. *Phys. Rev. Lett.*, 38(6):292–295.
- Omura, Y., Ninomiya, K., Umeda, T., Heikkila, W., and Matsumoto, H. (2008). Non-linear evolution of buneman instability. Downloaded November 2008 from <http://center.stelab.nagoya-u.ac.jp/web1/causes/2004/sm0009/>.
- Pécseli, H. (2006). *Waves and Oscillations in Plasmas*. Institute of Physics, University of Oslo, Norway. Lecture material for introductory course in Plasma Physics at the University of Oslo.
- Pécseli, H. L., Trulsen, J., and Armstrong, R. (1984). Formation of ion phase-space vortexes. *Phys. Scripta*, 29:241–253.
- Raadu, M. A. (1989). The physics of double layers and their role in astrophysics. *Physics Reports*, 178:25–97.
- Saeki, K., Michelsen, P., Pécseli, H. L., and Rasmussen, J. J. (1979). Formation and coalescence of electron solitary holes. *Phys. Rev. Lett.*, 42:501–504.
- Schamel, H. (1986). Electron holes, ion holes and double layers Electrostatic phase space structures in theory and experiment. *Physical Review*, 140:161–191.

- Schönhuber, M. J. (1958). *Zur weiteren Klärung innerer Vorgänge in Quecksilber-Niederdruckgasentladungen, insbesondere über das Auftreten und Verhalten von Doppelschichten (Broschiert)*. Hochschulbuchhandlung Lachner, München, Deutschland.
- Singh, N. (1980). Computer experiments on the formation and dynamics of electric double layers. *Journal of Plasma Physics*, 22:1–24.
- Smith, R. A. (1982). A review of double layer simulations. *Physica Scripta*, 2:238–251.
- Smith, R. A. (1982). Vlasov simulation of plasma double layers. *Physica Scripta*, 25:413–415.
- Tang, W. and Chan, V. (2005). Advances and challenges in computational plasma science. *Plasma Phys. and Control. Fusion*, 47:R1–R34.
- Trulsen, J. K. (2005). *Introduction to Numerical Simulation of Many-Particle Systems, Methods and Applications*. Institute of Theoretical Astrophysics, University of Oslo, Norway. Lecture material for advanced course in numerical methods at the Institute of Theoretical Astrophysics, University of Oslo.
- Wilhelmsson, H. (2000). *Fusion. A Voyage Through the Plasma Universe*. Institute of Physics Publishing, Bristol and Philadelphia.
- www.efda.org (2008). European fusion development agreement - wendelstein 7-as. url: [http://www.efda.org/eu\\_fusion\\_programme/machines-wendelstein\\_7as\\_d.htm](http://www.efda.org/eu_fusion_programme/machines-wendelstein_7as_d.htm), downloaded 1.12.2008.
- www.iter.org (2008). The iter project. url: <http://www.iter.org/>, downloaded 1.12.08.



# Appendix A

## Source code

In this appendix, the source code of the one dimensional double layer simulation program is included. The code was written in the programming language C++ and consist of the following files: **kode.h**, **main.cpp**, **1D\_plasma\_simulations.cpp**, **initializeDL.cpp**, **phi\_integrals.cpp**, **flux.cpp**, **fourier.cpp** and **diagnostics.cpp**. In addition to the main program, the executable file **crun** which compiles and executes the main program with the designated options. Each of these files are found in appendix A.1

The program import two libraries, both courtesy of Prof. Jan Trulsen at the Institute of Theoretical Astrophysics, University of Oslo, Norway. These libraries contain implementations of the Inversion method (see [Trulsen, 2005] and section 4.4.1) used during initialization of particles, and the Multigrid method Poisson solver (see section 4.5.2). These libraries are not included in this appendix.

The program requires a certain folder structure to work. All .cpp files are placed in the top-level folder `./`. Inside the top level folder, the subfolders `./diag`, `./log`, `./gnuplot`, `./gnuplot/marks` and `./plot` must be created. The two most crucial folders, `./diag` and `./log`, are automatically created and the contents of `./diag` and `./gnuplot/marks` are emptied upon execution of **crun**.

The program is started with the script **crun**. The most important compile option of the program is `-DBC_LR`, where *L* and *R* represent the boundary condition of the potential at respectively left and right boundary. *L* and *R* are replaced with either *D* for Dirichlet or *vN* for von Neumann type boundary conditions. If no boundary condition is give upon execution, the default option `-DBC_DvN` is chosen.

Plots of the results are made with *Gnuplot 4.2*. Each figure used in this thesis (unless stated otherwise) is made with the Gnuplot scripts listed in appendix A.2. Each of these scripts are placed in the folder `./gnuplot`. To generate the figure with the given scripts, enter either of the commands inside of the `./gnuplot` folder: "gnuplot results.p" or "gnuplot results.eps.p". Which command you choose depends if you want the output in one single .ps file, or multiple .eps files. The respective .ps or .eps files are placed in the folder `./plot`.

## A.1 Simulation code

### A.1.1 Compilation and execution

```
#!/bin/sh

#Deleting old contents of datafolders
if test ! -e diag; then
    mkdir diag
fi
if test ! -e log; then
    mkdir log
fi
if test ! -e gnuplot; then
    mkdir gnuplot
fi
if test ! -e gnuplot/marks; then
    mkdir gnuplot/marks
fi
if test ! -f gnuplot/marks/time0; then
    rm gnuplot/marks/*
fi

# set log file and date file
DATE=$(date +%Y.%m.%d-%H.%M.%S)
FILE="log/$DATE.simlog"

#Fastest fourier transform in the west compiler options
#FFTWS="-lfftws3 -lm"
FFTWS=""

#Compiling
g++ $FFTWS $SCRUNVALGRINDCOMPILE $1 $2 $3 $4 $5 $6 $7 $8 $9
    $SCRUNPROFILECOMPILE | tee $FILE

#Executing and writing to logfile
echo "Simulation starts at $(date)" | tee
    -a $FILE
$SCRUNVALGRIND ./a.out | tee
    -a $FILE
echo "Simulation ends at $(date)" | tee
    -a $FILE
echo $DATE > diag/date.dat

#Copying logfile
echo " Output copied to $FILE"
cp $FILE plot/00simlog.txt
```

### A.1.2 kode.h

```
/* ID Double Layer Simulations
 * Author : Vegard Lundby Rekaa
 * University of Oslo, Norway
 * March, 2009
 */

#include <cstdlib>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <cstring>
// #include <time.h>
// #include <sys/time.h>
#include <ctime>
using namespace std;

#define pi 3.14159265359
/*
// PRIMEROOT RANDOM NUMBER, WOJCIECH
#define BUCKETSIZ 1000
double primeroobucket[BUCKETSIZ];
double primerootno;
#ifdef DRAND48
inline double primeroot(void){
    double r;
    double sqrt_two=sqrt(2.0);
    int draw=(int)floor((drand48()*BUCKETSIZ));
    r=primeroobucket[draw];
    primerootno=primerootno+sqrt_two-floor((primerootno+
        sqrt_two));
    primeroobucket[draw]=primerootno;
    return r;
}
#else
double primeroot(void){return drand48();}
```

```
#endif
void init_primeroot(double seed){
    int i;
    double x,sqrt_two;
    sqrt_two=sqrt(2);
    srand48(0);
    x=seed;
    for(i=0; i<BUCKETSIZ; i++){
        x=x+sqrt_two-(int)(x+sqrt_two);
        primeroobucket[i]=x;
    }
    primerootno=x;
}
*/

double sign(double x){return x/abs(x);}
void lowpassfilter(double f[], int N){
    for(int i=1; i<N-1; ++i) f[i]=0.25*f[i-1]+0.5*f[i]
        +0.25*f[i+1];
}

void ErrorMessage(string text){
    cout << text << endl;
    exit(1);
}

/*
// Box-Mueller algorithm for normal distributed
// numbers
// N(mu = 0, sigma = 1)
double BoxMuller(){
    static int FIRST = 1;
    static double NEXT;
    double v1, v2, sqr, fac;

    if (FIRST) {
        do {
            v1 = 2.*drand48() - 1.;
            v2 = 2.*drand48() - 1.;
            sqr = v1*v1 + v2*v2;
        } while (sqr >= 1. || sqr == 0.);

        fac = sqrt(-2.*log(sqr)/sqr);
        NEXT = v2*fac;
        FIRST = 0;
        return v1*fac;
    } else {
        FIRST = 1;
        return NEXT;
    }
}
*/

/* *****
 * ERROR FUNCTION WITH SUBROUTINES
 * *****
 */
Error function
 * Source: Numerical Recopies in C.
 * Changed: float to double declarations
 */

#include <math.h>
//Returns the errorfunction
double erf(double x){
    double t,z,ans;

    z=fabs(x);
    t=1.0/(1.0+0.5*z);
    ans=t*exp(-z*z-1.26551223+t*(1.00002368+t*(0.37409196+
        t*(0.09678418+
        t*(-0.18628806+t*(0.27886807+t*(-1.13520398+t
            *(1.48851587+
            t*(-0.82215223+t*0.17087277)))))))));
    return x >= 0.0 ? 1.0-ans : ans-1.0;
}

//Returns the complementary error function
double erfc(double x){
    double t,z,ans;

    z=fabs(x);
    t=1.0/(1.0+0.5*z);
    ans=t*exp(-z*z-1.26551223+t*(1.00002368+t*(0.37409196+
        t*(0.09678418+
        t*(-0.18628806+t*(0.27886807+t*(-1.13520398+t
            *(1.48851587+
            t*(-0.82215223+t*0.17087277)))))))));
    return x >= 0.0 ? ans : 2.0-ans;
}

#include <math.h>
#define IMAX 100 //Maximum allowed number of iterations.
#define EPS 3.0e-7 //Relative accuracy.
#define FPMIN 1.0e-30 //Number near the smallest representable

/* Numerical Recipes standard error handler */
void nrerror(char error_text[]){
```

```

    fprintf(stderr, "Numerical_Recipes_run-time_error...\n"
    );
    fprintf(stderr, "%s\n", error_text);
    fprintf(stderr, "...now exiting_to_system...\n");
    exit(1);
}

//Metropolis algorithms.
//Written by Jan K. Trulsen, UiO in 2007 as an aid to my
//masters thesis.
//Source: Trulsen's lecture notes on statistics in numerical
//analysis
//Some adjustments made by me to fit my DL simulations (master
//thesis).
//
//      L. Rekaa
void Metropolis(double n[], int N, double *x, double *f,
               double delta, double a, double
               b){
    // Metropolis algorithm for a bounded domain (a, b)
    // pdf(x) : desired (unnormalized) probability density
    function
    // x      : input previous draw, output present draw
    // f      : f = pdf(x) at both input and output
    // delta  : random walk step length
    // Initialize new series by inputing f = 0 and x near
    // max of pdf()

    double xt, ft;
    int i;
    double w;

    xt = *x + delta*(2.*drand48() - 1.);
    if (xt < a) xt = 2.*a-xt; // mirroring at lower
    // boundary
    if (xt > b) xt = 2.*b-xt; // mirroring at upper
    // boundary

    w=x*N;
    i=int(floor(w));
    w=w-i;

    ft = n[i]*(1.-w) + n[i+1]*w;
    if ( ft > *f || ft >= *f*drand48() ) {
        *x = xt, *f = ft;
    }
}

void Metropolis(double (*pdf)(double,double,double), double u,
               double s,
               double *x, double *f, double delta, double a,
               double b){
    // Metropolis algorithm for a bounded domain (a, b)
    // pdf(x) : desired (unnormalized) probability density
    function
    // x      : input previous draw, output present draw
    // f      : f = pdf(x) at both input and output
    // delta  : random walk step length
    // Initialize new series by inputing f = 0 and x near
    // max of pdf()

    double xt, ft;

    xt = *x + delta*(2.*drand48() - 1.);
    if (xt < a) xt = 2.*a-xt; // mirroring at lower
    // boundary
    if (xt > b) xt = 2.*b-xt; // mirroring at upper
    // boundary

    ft = pdf(xt, u, s);
    if ( ft > *f || ft >= *f*drand48() ) {
        *x = xt, *f = ft;
    }
}

void Metropolis(double (*pdf)(double, double), double u,
               double *x,
               double *f, double delta, double a){
    // Metropolis algorithm for a bounded domain (a, b)
    // pdf(x) : desired (unnormalized) probability density
    function
    // x      : input previous draw, output present draw
    // f      : f = pdf(x) at both input and output
    // delta  : random walk step length
    // Initialize new series by inputing f = 0 and x near
    // max of pdf()

    double xt, ft;

    xt = *x + delta*(2.*drand48() - 1.);
    if (xt < a) xt = 2.*a-xt; // mirroring at lower
    // boundary

    ft = pdf(xt, u);
    if ( ft > *f || ft >= *f*drand48() ) {
        *x = xt, *f = ft;
    }
}

```

## A.1.3 main.cpp

```

/*      ID Double Layer Simulations
 *      Author : Vegard Lundby Rekaa
 *      University of Oslo, Norway
 *      March, 2009
 */

#define XDIM 1
#ifdef BC_DD
    #define BCX0 0
    #define BCX1 0
#endif
#ifdef BC_vND
    #define BCX0 1
    #define BCX1 0
#endif
#ifdef BC_vNvN
    #define BCX0 1
    #define BCX1 1
#endif
#ifdef BC_vNvN
    #ifndef BC_vND
        #ifndef BC_DD
            #define BCX0 0
            #define BCX1 1
        #endif
    #endif
#endif
#ifdef BC_vND
    #ifndef BC_DD
        #define BCX0 0
        #define BCX1 1
    #endif
#endif
#endif

double me, mp, Qp, Qe, QMe, QMp, vthe, vthp, drifte, driftp;
long Ne, Np, particles;
double L, lx, ux, dt;
double *xe, *xp, *ve, *vp, *dve, *dvp, phi0, phiDL, BCphi0,
        BCphiDL;
double flux_ea, flux_erd, flux_ia, flux_ird;
int N_time;

//Headerfile
#include "kode.h"
//Jans contribution
#include "Utils.cpp"
#include "MultiGrid.cpp"
//My files
#include "phi_integrals.cpp"
#include "waterbag.cpp"
#include "initializeDL.cpp"
#include "TVhist.cpp"
#include "ID_plasma_simulations.cpp"
//include "influzhist.cpp"
#include "flux.cpp"
#include "diagnostics.cpp"
#include "tracing.cpp"
#include "fourier.cpp"
#include "presentation.cpp"

void set_globals(int *N){

    double TionTe, mach, space, P, Adx, Avdtdx, time,
           num_part;

    //Variable paramters
    L=250; //Spatial range
    phiDL=200.0; //Potential leap (must be >0)
    mp=1.; //Ion mass
    mach=5; //Driftvelocity

    //For the scriptfile BCcopy
    double temp_L, temp_phiDL, temp_mp, temp_mach;
    temp_L=0;
    temp_phiDL=0;
    temp_mp=0;
    temp_mach=0;

    particles=long(1e7);
    time=200.; //Simulation time in units of
    // dt

    num_part=1e5; //Temporary number for no. of
    // particles

    // "SIZES" OF SIMULATION
    space=250; //Size of plasma added
    Adx=0.5; //Accuracy x
    Avdtdx=0.6; //Accuracy vdt/dx

    //PHYSICAL QUANTITIES
    me=1.0; //Electron mass
    TionTe=1.0; //Temperature ratio
    phi0=0.0;

    //
    // DO NOT CHANGE BEYOND THIS LINE
    // Total simulation time
    // N_time=int(pow(2.,P))-1; //Must be >=10 !
    // Position boundaries

```

```

    lx=0.-space;          ux=L+space;
    //Charge
    //Code is hardcoded on the assumption of negative
    //electrons and positive ions. Do not change!
    Qp=1./num_part;      Qe=-1./num_part;
    //Mass
    mp=num_part;          me/=num_part;
    //Mass charge ratio
    QMp=Qe/me;            QMp=Qp/mp;
    //Thermal velocity
    vthe=1.0;             vthp=sqrt(TionTe*me/mp);
    //Driftvelocity
    //Ion drift must be negative, electron drift must be
    //positive
    drifte=mach*vthe;     driftp=-mach*vthp;
    //If Dirichlet boundary conditions, these are needed
    BCphi0=phi0;          BCphiDL=phiDL;
    //Automatic set spatial resolution
    double dx=1.;
    for(double p=7.; dx>Adx; ++p){
        *N=int(pow(2.,p))+1;
        dx=(ux-lx)/((N)-1);
        //cout<<"dx="<<dx<<" p="<<p<<" N="<<N<<endl;
    }
    //Automatic set temporal resolution
    double vth=vthe; if(vthp>vth) vth=vthp;
    dt=Avdtdx*dx/((mach+2.)*vth);
    double vtdtx=vth*(mach+2.)*dt/dx;

    N_time=int(time/dt);

/*
    double t=0.;
    for(double p=8.; t<time && p<14.; ++p){
        N_time=int(pow(2.,p))-1;
        t=N_time*dt;
        //cout<<"time: "<<t<<" "<<N_time<<" "<<p<<" "<<
        dt<<endl;
    }
    N_time=int(N_time);
*/
    void print_simdetails(double, int&);
    void initialstabilitycheck(double, double, double, int
    &);
    print_simdetails(vtdtx,*N);
    initialstabilitycheck(vtdtx,mach, TionTe, *N);

}

void finalize(bool end){

    finalizemother();
    printTVhist(N_time);

    if(end) exit(1);

}

int main (int argc, const char* argv[] ) {

    /*
    *      INITIALIZATION
    */
    system("rm_diag/*.dat"); //Removing old data
    srand48(2); //Initializing drand48
    int N; //Size of spatial mesh
    set_globals(&N); //Set global variables

    double RHO[N], PHI[N], E[N+1]; //Fields
    long temp=initDL(false); //Calc. auto part
    //numbers
    reweight(temp); //Adjusting part.
    //number
    temp=initDL(true); //Initializing Double
    //Layer

    initialdiagnostics(PHI,N_time, N); //
    //Diagnostics;

    //MG poisson solver initialization
    int n[XDIM]; double h[XDIM];
    initMGpoisson(n,h,N);
    MultiGrid *mg = new MultiGrid(n,h);

    //Div. test procedures
#ifdef TANHPHI
    void tanhphi(); tanhphi(); //in waterbag.cpp
#endif
#ifdef TESTMOTHER
    void testmother(); testmother();
#endif
#ifdef INIT
    cout << "Checked_initialization ,_exiting."<<endl<<endl
    ;
    exit(0);
#endif

    /*
    *      TEMPORAL SIMULATIONS
    */
    cout<<endl<<"-----STARTING_TEMPORAL_
    SIMULATIONS-----"<<endl;
    //Zeroth timestep

```

```

    charge(RHO,N); //Assign particles to
    //spacegrid
    poissonMG(PHI,RHO,N,mg,n,h); //Find ES potential
    Efield(E,PHI,N); //Find Electric field
    deltaV(E,N); //Find change in
    //velocity
    init_leapfrog(); //Backstepping
    //velocities to t=-1/2

    diagnostics(RHO,PHI,E,N, 0); //Diagnostics
    double phidl[N_time+1]; phidl[0]=phiDL; //Diagnostics

    //Timeloop start
    for(int t=1; t<N_time+1; ++t){

        deltaV(E,N); //Find change
        //in vel of part
        leapfrog(t); //Stepping pos
        //and vel
        flux(t); //Flux of
        //through boundary

        charge(RHO,N); //Assign part
        //to spacegrid
        poissonMG(PHI,RHO,N,mg,n,h); //Find ES
        //potential
        Efield(E,PHI,N); //Find
        //Electric field

        phidl[t]=PHI[N-1]-PHI[0]; //Diagnostics
        //trace(xp,vp,t); //Diagnostics
        diagnostics(RHO,PHI,E,N,t); //Diagnostics

    }

    FFTphidl(phidl,N_time+1,dt);

    //
    mg->MultiGrid();
    finalize(false);

    cout<<"-----SIMULATIONS_DONE-----"
    << endl;
    return 0;
}

```

## A.1.4 1D\_plasma\_simulations.cpp

```

/*      1D Double Layer Simulations
*      Author : Vegard Lundby Rekaa
*      University of Oslo, Norway
*      March, 2009
*/

void charge(double Qcell[], int N){
    //Applying the particle-in-cell (PIC) scheme to the
    //particles
    //position. All continuous positions are related to a
    //grid point
    //through the PIC scheme Cloud-in-cell. Normalization
    //is included
    //the factor (dx)^2 wich is not a part of the charge
    //density. I am
    //doing this to save simulation time by not adding
    //them later in
    //my poisson solver.

    int index;
    double x;
    double dx=(ux-lx)/(N-1);

    for(int i=0; i<N; ++i) Qcell[i]=0.0;

    //Electrons
    for(int i=0; i<Ne; ++i){
        x=(xe[i]-lx)/dx;
        index = int(x); //index=grid point adress
        //x is now the distance to the nearest lower
        //gridpoint.
        //a number between 0 and 1 where 1 is the
        //distance
        //between gridpoints
        x-=index;

        //Assigning a particles charge as it was a
        //cloud to points
        Qcell[index]+=(1.0-x)*Qe;
        Qcell[index+1]+=x*Qe;
    }
    //Protons (see comments above)
    for(int i=0; i<Np; ++i){
        x=(xp[i]-lx)/dx;
        index = int(x);
        x-=index;

        Qcell[index]+=(1.0-x)*Qp;
    }
}

```



```

        Qcell[index+1]+=x*Qp;
    }
    //Boundary conditions
    Qcell[0]*=2.0;      Qcell[N-1]*=2.0;

    for(int i=0; i<N; ++i) Qcell[i]/=dx;

    lowpassfilter(Qcell,N);
}

void initMGpoisson(int n[], double h[], int N){
    double dx=(ux-lx)/(N-1);
    for(int i=0; i<XDIM; ++i) n[i]=N, h[i]=dx;
    //Assuming XDIM == 1
}

void poissonMG(double PHI[], double RHO[], int N, MultiGrid *
mg,
int n[], double h[]){
    //Assuming XDIM == 1
    double rhoMG[n[0]+2], phiMG[n[0]+2];
    //Copying RHO to array of n[0]+2 dim and reversing
    sign
    for(int i=1; i<n[0]+1; ++i) rhoMG[i]=-RHO[i-1];

    //Boundary conditions
    #if BCX0 == 0 //If phi(x=0) : Dirichlet BC
    rhoMG[0]=BCphi0; //phi=0
    #endif
    #if BCX0 == 1 //If phi(x=0) : von Neuman
    rhoMG[0]=0.; //dphi/dx=0
    #endif
    #if BCX1 == 0 //If phi(x=L) : Dirichlet BC
    rhoMG[n[0]+1]=BCphiDL; //phi=phiDL
    #endif
    #if BCX1 == 1 //If phi(x=L) : von Neuman
    rhoMG[n[0]+1]=0.; //dphi/dx=0
    #endif

    (*mg).mglin(phiMG, rhoMG, n, h);
    (*mg).delbc2rho(rhoMG, n, h);

    //Copy back to PHI-array
    #ifdef BC_vNvN
    double phimid=0.0;
    phimid=phiMG[n[0]/2]-BCphiDL/2.;
    for(int i=1; i<n[0]+1; ++i) PHI[i-1]=phiMG[i]-phimid;
    #else
    for(int i=1; i<n[0]+1; ++i) PHI[i-1]=phiMG[i];
    #endif
    phi0=PHI[0];      phiDL=PHI[N-1];
    if(isnan(phi0)) cout << "phi0_is_NaN" << endl;
    if(isnan(phiDL)) cout << "phiDL_is_NaN" << endl;
}

void Efield(double E[], double PHI[], int N){
    //Find the electric field with the first derivative of
    the
    //potential. E is calculated as a midpoint in i+1/2
    not in
    //i compared to the grid of Qcell and phi. Notice that
    E is
    //a matrix with two more elements than needed to fill
    the
    //midpoints mentioned above. It is so to have the
    boundaries
    //in the arrays first and last element.

    double dx=(ux-lx)/(N-1);

    for(int i=1; i<N; ++i) E[i]=-(PHI[i]-PHI[i-1])/dx;

    //Boundary conditions
    //Extrapolating E's outside of the domain
    E[0]=2*E[1]-E[2];
    E[N]=2*E[N-1]-E[N-2];
}

void deltaV(double E[], int N){
    //From the electric field I find the acceleration
    multiplied with
    //the timestep (wich means this is the change in
    velocity pr
    //timestep. Electric field is mapped back to particle
    position
    //in the same fassion the particles positions where
    mapped onto
    //the grid earlier. The procedures are still different
    since
    //E is calculated in midpoints (i+1/2).

```

```

    int index;
    double x;
    double dx=(ux-lx)/N;
    double dv_factor_p=dt*QMp; //Velocitystep,
    protons
    double dv_factor_e=dt*QMe; //Velocitystep
    electrons

    //Electrons (for comments on procedure, see "void PIC
    ())"
    for(int i=0; i<Ne; ++i){
        x=(xe[i]-lx)/dx-0.5;
        index = int(floor(x))+1;
        x=index;

        //CIC weighting of Efields in gridpoints to
        particles pos.
        dve[i]=(E[index]*(1.0-x)+E[index+1]*x)*
        dv_factor_e;
    }

    //Protons
    for(int i=0; i<Np; ++i){
        x=(xp[i]-lx)/dx-0.5;
        index = int(floor(x))+1;
        x=index;

        //CIC weighting of Efields in gridpoints to
        particles pos.
        dvp[i]=(E[index]*(1.0-x)+E[index+1]*x)*
        dv_factor_p;
    }
}

void init_leapfrog(){
    //Backstepping one half timestep (Leapfrog)

    for(int i=0; i<Ne; ++i)
        ve[i] = dve[i]/2.0;

    for(int i=0; i<Np; ++i)
        vp[i] = dvp[i]/2.0;
}

void leapfrog(int time){
    //Calculating v(t+dt/2) and x(t+dt) (leapfrog)
    //Leapfrog: velocities found in midpoints between
    //those of positions and accelerations
    for(long i=0; i<Ne; ++i){
        ve[i]+=dve[i];
        xe[i]+=ve[i]*dt;
    }

    for(long i=0; i<Np; ++i){
        vp[i]+=dvp[i];
        xp[i]+=vp[i]*dt;
    }
}

```

## A.1.5 initializeDL.cpp

```

/*      1D Double Layer Simulations
 *      Author : Vegard Lundby Rekaa
 *      University of Oslo, Norway
 *      March, 2009
 */

void reweight(long temp){
    double f=double(temp)/(double(particles));
    Qp*=f; Qes*=f; mes*=f; mp*=f;
}

void rhoofphi(double rho[], double alpha[], int N){
    //Returns the charge density as a function of
    potential

    double nea(double);
    double ner(double);
    double ned(double);
    double nia(double);
    double nir(double);
    double nid(double);

```

```

double p, rsum=0.0;
double d_phi=phiDL/(N-1);
for(int i=0; i<N; ++i){
    p=i*d_phi;
    rho[i]=Qp*(alpha[2]*nia(p) + alpha[3]*nir(p) +
        alpha[3]*nid(p))
        + Qe*(alpha[0]*nea(p) + alpha[1]*ner(p)
        ) +
        alpha[1]*ned(p));
    rsum+=rho[i];
}
rsum*=d_phi;

//Diagnostics
double max=0.0, r, absint=0.0; //Find maximum rho
value!
for(int i=0; i<N; ++i){
    r=abs(rho[i]);
    if(r>max)max=r;
    absint+=r;
}
absint*=d_phi;

//bc0: Charge density at x=0 taken to max charge
density in DL
//bcL:  $\frac{\text{max}}{\text{max}}$  x-L
//bc:  $\frac{\text{max}}{\text{max}}$  Integrated charge density
double bc0=rho[0]/max*100.;
double bcL=rho[N-1]/max*100.;
double bc=rsum/absint*100.;

//Print results to screen
cout << "Charge_density" << endl;
cout << "Overall_neutrality=" << bc << "%" << endl;
cout << "Neutrality_at_boundaries=" << bc0 << "%" << bcL << "%" << endl;
//If essential boundary conditions are violated, print
warning!
if((bc0*bc0>20. || bcL*bcL>20. || bc*bc>20.){
    cout
    << "*****" << endl
    << "CHARGE_DENSITY_BC_VIOLATED!!!" << endl
    << "*****" << endl;
}
}

void sagdeev(double alpha[], double phi[],
    double (*phifunction)(double), int N){
    //Through charge density and the Sagdeev potential,
    //calculate the last alpha so that the amplitude of
    the DL
    //charge density matches the DL potential leap phiDL.
    //Also sets the final potential profile phi(x) wich is
    stored in
    //global phi_array accessible through function "double
    phi(x)".
    //Returns the array alpha after adjusted to its final
    values.

    //Finding rho(phi)
    double rho[N]; rhoofphi(rho, alpha, N);

    //Finding the Sagdeev potential V(phi)
    //from the charge density
    double V[N];
    double d_phi=phiDL/(N-1);
    V[0]=0.;
    for(int i=1; i<N; ++i){
        V[i] = V[i-1] - 0.5*(rho[i-1]+rho[i])*d_phi;
        if(V[i]*V[i]<1e-14 && V[i]<0.0) V[i]=-V[i];
    }
    for(int i=1; i<N; ++i) //Test if V[i]<0. If so, print
    if(V[i]<=0.0){ //warning and exit program
        cout << "*****ERROR*****Wrong_Sagdeev_
        potential:" << endl;
        cout << V[i] << " " << rho[i] << " " << endl;
        i << endl;
        cout << "Forcing_exit!!" << endl << endl;
        exit(0);
    }
}

//Calculate alpha raising the magnitude of the DL
charge density
//to a level matching phiDL.
double I[N];
I[0]=0.0; //Special treatment of singularity
I[1]=2.*sqrt(-d_phi/.5/rho[1]);
if(isnan(I[1])){
    cout << "*****I[1]_is_Nan!!!!" << endl;
}

    exit(0);
}
for(int i=2; i<N-1; ++i) I[i]=I[i-1] + d_phi/sqrt(V[i]);
//Loop
I[N-1]=I[N-2] + 2.*sqrt(d_phi/.5/rho[N-2]);
//Boundary
double alpha_s=0.5*I[N-1]*I[N-1]/L/L;
//Result

//find x(phi)
double x[N];
double a=1./sqrt(2.*alpha_s);
x[0]=0.;
for(int i=0; i<N-1; ++i) x[i]=a*I[i];
x[N-1]=L;

//inverting to phi(x)
phi[0]=0.;
int j;
double w, xi;
double dx=L/(N-1);
j=1;
for(int i=1; i<N; ++i){
    xi=i*dx;
    //j=0;
    while(xi>x[j])++j;
    if(j>N-1) cout << "j_too_large:" << j
    << endl;

    w=(xi - x[j-1])/(x[j] - x[j-1]);
    phi[i] = (j-1)*d_phi + w*d_phi;

    if(phi[i]<0.0 || phi[i]>phiDL){
        cout << "INCORRECT_phi[i]=" << phi[i] <<
        endl;
        cout << "Forcing_exit!" << endl;
        exit(0);
    }
}

//Returnvalues
for(int i=0; i<4; ++i) alpha[i]=alpha_s;

//For diagnostics: find rho(x)
double rhox[N], p;
for(int i=0; i<N; ++i){
    p=(*phifunction)(i*dx);
    rhox[i]=Qp*(alpha[2]*nia(p)+alpha[3]*nir(p)+
        alpha[3]*nid(p))
        +Qe*(alpha[0]*nea(p)+alpha[1]*ner(p)+
        alpha[1]*ned(p));
}

//Diagnostics
ofstream sag("diag/supermethod.dat");
for(int i=0; i<N; ++i) sag << i*dx << " "
    << i*d_phi << " "
    << V[i] << " "
    << x[i] << " "
    << phi[i] << " "
    << rho[i] << " "
    << rhox[i] << " "
    << I[i] << " "
    << endl;
sag.close();
}

void BC_ChargeNeutrality(double alpha[]){
    double nea(double);
    double ner(double);
    double ned(double);
    double nia(double);
    double nir(double);
    double nid(double);

    double alpha_ir=1.0, alpha_er=1.0;
    double alpha_ea=1.0, alpha_ia=1.0;

    int N=200;
    double d_phi=phiDL/(N-1);

    //Integrated density over space for all six sorts
    double Nea=0.0, Ner=0.0, Ned=0.0, Nia=0.0, Nir=0.0,
    Nid=0.0;
    for(double p=0.; p<phiDL; p+=d_phi){
        Nea+=nea(p)*d_phi;
        Ner+=ner(p)*d_phi;
        Ned+=ned(p)*d_phi;
        Nia+=nia(p)*d_phi;
        Nir+=nir(p)*d_phi;
        Nid+=nid(p)*d_phi;
    }

    //Density of the six particle sorts taken at
    boundaries
    double nea_0=nea(0);

```

```

double nea_L=nea(phiDL);
//ner_0=0
double ner_L=ner(phiDL);
double ned_0=ned(0);
double ned_L=ned(phiDL);
double nia_0=nia(0);
double nia_L=nia(phiDL);
double nir_0=nir(0);
//nir_L=0
double nid_0=nid(0);
double nid_L=nid(phiDL);

double aea=1.0, aer, aia, air;
//alpha of reflected electrons
acr=aea*(
    (Nea*nia_L/Nia-nea_L) +
    (nea_0-Nea*nia_0/Nia)*(nid_L-(Nir+Nid)
    *nia_L/Nia)
    /(nir_0+nid_0-(Nir+Nid)*nia_0/Nia)
    )/(
    (ner_L+ned_L-(Ner+Ned)*nia_L/Nia) -
    (ned_0-(Ner+Ned)*nia_0/Nia)
    *(nid_L-(Nir+Nid)*nia_L/Nia)
    /(nir_0+nid_0-(Nir+Nid)*nia_L/Nia)
    );

//alpha of reflected ions
air=(aer*(ned_0-(Ner+Ned)*nia_0/Nia)+aea*(nea_0-Nea*
    nia_0/Nia)
    /(nir_0 + nid_0-(Nir+Nid)*nia_0/Nia));
//alpha of accelerated ions
aia = (aea*Nea + aer*(Ner+Ned) - air*(Nir+Nid))/Nia;

//Storing alpha results in return vector alpha
alpha[0]=aea; alpha[1]=aer; alpha[2]=aia; alpha
[3]=air;

//Diagnostics
for(int i=0; i<4; ++i) if(alpha[i]<0.){
    cout << "ERROR:_NEGATIVE_PARTICLE_NUMBERS!!
    "
    <<"_Forcing_exit!"<<endl<<endl;
    exit(0);
}

}

void fill(long *Npart, double x[], double v[],
    long Nfill, double temp_x[], double temp_v[]){

    for(long i=0; i<Nfill; ++i){
        x[*Npart+i] = temp_x[i];
        v[*Npart+i] = temp_v[i];
    }
    *Npart+=Nfill;
}

void partlenumbers(double alpha[], long N[]){
    //Calculating the number of particles of specie 's' to
    be
    //generated from N_s=|int_0^phiDL alpha_s n_s(phi)
    d_phi

    double nea(double);
    double ner(double);
    double ned(double);
    double nia(double);
    double nir(double);
    double nid(double);

    double ea=0., er=0., ed=0, ia=0., ir=0., id=0.;
    double dx=(ux-lx)/300;
    double p;
    //Integral of density functions for the six species
    for(double x=lx; x<=ux; x+=dx){
        p=phi(x);
        ea+=nea(p)*dx;
        er+=ner(p)*dx;
        ed+=ned(p)*dx;
        ia+=nia(p)*dx;
        ir+=nir(p)*dx;
        id+=nid(p)*dx;
    }

    //Multiplying with alpha to get particle numbers
    N[0]=long(alpha[0]*ea); //Accelerated electrons
    N[1]=long(alpha[1]*er); //Reflected electrons
    N[2]=long(alpha[1]*ed); //Decelerated electrons

    N[3]=long(alpha[2]*ia); //Accelerated ions
    N[4]=long(alpha[3]*ir); //Reflected ions
    N[5]=long(alpha[3]*id); //Decelerated ions

    //Global variables telling the number of particles
    generated
    //at each moment. Naturally they are equal to 0 now.
    Ne=0; Np=0;

//Diagnostics output
cout <<"_Particle_numbers"<< endl
<<"_Nea="<<N[0]<<"_Ner="<<N[1]<<"_Ned="<<N
[2]
<<"_Nia="<<N[3]<<"_Nir="<<N[4]<<"_Nid="<<N
[5]
<<endl
<<"_Ne="<<N[0]+N[1]+N[2]<<"_Ni="<<N[3]+N
[4]+N[5]
<<"_Ni-Ne="<<(-(N[0]+N[1]+N[2])+(N[3]+N[4]+N
[5]))
<<"_((Ni-Ne)/(avg(N)))="
<<double(-(N[0]+N[1]+N[2])+(N[3]+N[4]+N[5]))
/ double((N[0]+N[1]+N[2])+(N[3]+N[4]+N
[5]))*50.
<< "%s"<< endl<< endl;
}

void allocate_xvarrays(long ne, long ni){
    //Allocating arrays to global pointers *xe,*ve,*xp,*vp
    ,*dve,*dvp
    //Stores positions, velocities and change in
    velocities for
    //electrons and protons

    //Setting size of arrays larger than the number of
    particles I'm
    //starting with, to allow a positive net flux of
    particles
    long alloc_e=long(ne*1.5);
    long alloc_i=long(ni*1.5);

    //Allocating
    xe = new double[alloc_e];
    ve = new double[alloc_e];
    xp = new double[alloc_i];
    vp = new double[alloc_i];
    dve = new double[alloc_e];
    dvp = new double[alloc_i];

    //Zeroing all elements of all arrays before use.
    for(long i=0; i<alloc_e; ++i){ xe[i]=0.0; ve[i]=0.0;
        dve[i]=0.0;}
    for(long i=0; i<alloc_i; ++i){ xp[i]=0.0; vp[i]=0.0;
        dvp[i]=0.0;}

}

void generate(long K[], double alpha[]){
    //From known particle distrobution functions f(x,v)
    use the
    //inversionmethod to decide how particles are
    distributed in
    //phasespace, then draw them after this pattern. Last
    step stores
    //all drawn particles in global arrays for position/
    velocity for
    //both particle species.

    double *x, *y, fnorm;

    long N=0; for(int i=0; i<6; ++i) if(K[i]>N) N=K[i];
    N=long(N+1);

    x = new double[N]; //Position temp array
    y = new double[N]; //Velocity temp array

    void ps_plot(string, double [], double [], long, int);
    const int res=300; //Space/velocity gridsize in
    inversion
    long start, stop;

    //For all six particle species, particles are drawn
    from
    //their respective PDF's, put into temporary arrays x
    and y,
    //then placed in common arrays for electrons and ions
    //respectively.
    cout<<"_Generating_particles" << endl;
    //ACCELERATED ELECTRONS
    cout<<"_Electrons:_" ;
    cout << "_Accelerated_" ;
    N=K[0];
    Inversion ea(lx,ux,res,lea(lx),uea(ux),res,lea,uea,fea
        ,fnorm);
    ea.draw(N, x, y);
    fill(&Ne,xe,ve,N,x,y);

    //REFLECTED ELECTRONS
    cout << "_Reflected_" ;
    N=K[1];
    Inversion er(lx,ux,res,ler(ux),uer(ux),res,ler,uer,fer
        ,fnorm);
    er.draw(N, x, y);
    fill(&Ne,xe,ve,N,x,y);

    //DECELERATED ELECTRONS
    if(K[2]!=0){

```

```

        cout << "_Decelerated" ;
        N-K[2]; start=stop; stop+=N;
        Inversion ed(lx,ux,res,led(ux),ued(lx),res,led
            ,ued,fed,fnorm);
        ed.draw(N, x, y);
        fill(&Ne,x,ve,N,x,y);
    }
    cout<<endl<<"_Ions:_";
    //ACCELERATED IONS
    cout << "Accelerated_";
    N-K[3];
    Inversion ia(lx,ux,res,lia(lx),uia(ux),res,lia,uia,fia
        ,fnorm);
    ia.draw(N, x, y);
    fill(&Np,xp,vp,N,x,y);

    //REFLECTED IONS
    cout << "_Reflected_";
    N-K[4];
    Inversion ir(lx,ux,res,lir(lx),uir(lx),res,lir,uir,fir
        ,fnorm);
    ir.draw(N, x, y);
    fill(&Np,xp,vp,N,x,y);

    //DECELERATED IONS
    if(K[5]!=0){
        cout << "_Decelerated";
        N-K[5];
        Inversion id(lx,ux,res,lid(ux),uid(lx),res,lid
            ,uid,fid,fnorm);
        id.draw(N, x, y);
        fill(&Np,xp,vp,N,x,y);
    }
    cout<<endl;
    delete x,y;
}

long initDL(bool gen){
    cout << endl<< "-----"
        << "___INITIALIZING_DL___" << endl;
    //Array N: number of particles
    // [0] : accelerated electrons
    // [1] : reflected electrons
    // [2] : decelerated electrons
    // [3] : accelerated ions
    // [4] : reflected ions
    // [5] : decelerated ions
    long N[6];

    //Array alpha: weighting of PDF's
    // [0] : accelerated electrons
    // [1] : reflected and decelerated electrons
    // [2] : accelerated ions
    // [4] : reflected and decelerated ions
    double alpha[4];

    //The boundarycondition of having overall charge
    //neutrality
    //and charge neutrality at the boundaries gives 3
    //equations
    //for solving the alphas, wich is done here
    BC_ChargeNeutrality(alpha);

    //Through a calculation of the Sagdeev potential,
    //the final criterium for alpha is laid and
    //the potential profile consistent with a
    //DL is found. After this, phi_array is accessible
    //through
    //the function "double phi(double x)".
    sagdeev(alpha, phi_array, phi, phi_array_length);

    //Calculating the expected influx number at both
    //boundaries for
    //both particle species. Set the value of the global
    //variables
    //flux_ea, flux_erd, flux_ia and flux_ird.
    void initflux(double []);
    initflux(alpha);

    //Integrating the particle densities with the
    //weightings alpha
    //to find the number of each particle specie. Also
    //setting global
    //variables Ne and Np (total electron and ion number)
    partlenumbers(alpha,N);

    if(gen){
        //Allocate the global arrays for
        //electron/ion position/velocity
        allocate_xarrays(N[0]+N[1]+N[2],N[3]+N[4]+N
            [5]);

        //Draw particles from PDF's,
        //in the amount described by N,
        //and place them in global position/velocity
        //arrays.
        generate(N,alpha);

```

```

        return Ne;
    }else{
        cout <<"_Adjusting_particle_numbers" << endl;
        return N[0]+N[1]+N[2];
    }
}

```

## A.1.6 phi\_integrals.cpp

```

/*      1D Double Layer Simulations
 *      Author : Vegard Lundby Rekaa
 *      University of Oslo, Norway
 *      March, 2009
 */

int phi_array_length=10000;
double *phi_array = new double[phi_array_length];
double phi(double x){

    double w;
    int m;
    //Linear interpolation
    w=x*phi_array_length/L;
    m=int(floor(w));
    w=m;

    if(m <= 0){
        return 0;
    }
    else if(m >= phi_array_length-1) return phiDL;
    else return (1-w)*(phi_array[m]) + w*(phi_array[m
        +1]);
}

//ELECTRON HELP FUNCTIONS
double fea(double x, double y){
    double a=y*y+2.*QMe*phi(x);
    if(a*a<1e-8 && a < 0.0)a=0.0;
    double u=sqrt(a);

    //return 1./sqrt(2.*pi)/vthe*exp(-.5*(u-drifte)*(u-
        drifte)/vthe/vthe);
    return exp(-.5*(u-drifte)*(u-drifte)/vthe/vthe);
}
double vfea(double v){
    return v*fea(0., v);
}
double uea(double x){
    double veinf2=pow(15.*vthe+drifte,2); //
    INFINITY
    return sqrt(veinf2-2.*QMe*phiDL);
}
double lea(double x){
    return sqrt(-2.*QMe*phi(x));
}

double fer(double x, double y){
    //return 1./sqrt(2.*pi)/vthe
    // *exp( (-.5*y*y + QMe*(phiDL-phi(x)))/vthe/vthe
        );
    return exp( (-.5*y*y + QMe*(phiDL-phi(x)))/vthe/vthe
        );
}
double vfer(double v){
    return abs(v)*fer(L, v);
}
double uer(double x){
    return sqrt(-2.*QMe*phi(x));
}
double ler(double x){
    return -sqrt(-2.*QMe*phi(x));
}

double fed(double x, double y){
    //return fer(x,y);
    return fer(x,y);
}
double ued(double x){
    return -sqrt(-2.*QMe*phi(x));
}
double led(double x){
    double veinf2=pow(15.*vthe+drifte,2); //
    INFINITY
    return -sqrt(veinf2-2.*QMe*phiDL);
}

```

```
// ION HELP FUNCTIONS
double fia(double x, double y){

    double a=y*y-2.*Qmp*(phiDL-phi(x));
    if(a*a<1e-8 && a < 0.0) a=0.0;
    double u=sqrt(a);
    //return 1./sqrt(2*pi)/vthp
    // *exp((-0.5*(u-driftp)*(u-driftp)/vthp/vthp));
    return exp((-0.5*(u-driftp)*(u-driftp)/vthp/vthp));
}
double vfia(double v){
    return abs(v)*fia(L, v);
}
double uia(double x){
    return -sqrt(2.*Qmp*(phiDL-phi(x)));
}
double lia(double x){
    double vpinf2=pow(15.*vthp-driftp,2); //
    INFINITY
    return -sqrt(vpinf2+2.*Qmp*phiDL);
}

double fir(double x, double y){
    //return 1./sqrt(2*pi)/vthp
    // *exp((-0.5*y*y - Qmp*phi(x))/vthp/vthp);
    return exp((-0.5*y*y - Qmp*phi(x))/vthp/vthp);
}
double vfir(
    double v){return v*fir(0., v);
}
double uir(double x){
    return sqrt(2.*Qmp*(phiDL-phi(x)));
}
double lir(double x){
    return -sqrt(2.*Qmp*(phiDL-phi(x)));
}

double fid(double x, double y){
    //return fir(x,y);
    return fir(x,y);
}
double uid(double x){
    double vpinf2=pow(15.*vthp+driftp,2); //
    INFINITY
    return sqrt(vpinf2+2.*Qmp*phiDL);
}
double lid(double x){
    return sqrt(2.*Qmp*(phiDL-phi(x)));
}

//INTEGRALS OF DENSITY
double nea(double phi){

    int it=0;

    double a=-2*phi*QMe;
    double dv=0.05*vthe;
    double n=0.0;
    double d=0.0;
    double v=dv;

    //ofstream fil("diag/nea.dat");
    //fil << n << " " << d << " " << v << endl;
    do{
        d=v*fea(0.,v)/sqrt(v*v+a);
        n+=d*dv;
        v+=dv;
        ++it;

        if(isnan(d)){
            cout << "nea_isnan: ";
            cout << fea(0.,v) << " "
                 << v*v+a << " "
                 << v*v << " "
                 << a << " "
                 << endl;
        }
        //fil << n << " " << d << " " << v << endl;

    }while(d>0.01*dv || it < 100);
    //fil.close();

    // cout << " nea it=" << it << " n=" << n << endl;
    return n;
}

double ned(double phi){

    int it=0;

    double v=sqrt(-2.*phiDL*QMe);
    double vs2=2*(phi-phiDL)*QMe;
    double dv=0.01*v;
    double d=0;
    double n=0.0;

    n=(3./2.)*sqrt(v*dv/2)*fed(L-v);
    v+=dv;

    //ofstream fil("diag/ned.dat");
    //fil << n << " " << d << " " << v << endl;
    do{
        d=v*fed(L-v)/sqrt(v*v - vs2);
        n+=d*dv;
        v+=dv;
        ++it;

        if(isnan(d)){
            cout << "ned_isnan: ";
            cout << fed(L-v) << " "
                 << v << " " << drifte << " "
                 << vthe << " "
                 << v*v-vs2 << " "
                 << v*v << " "
                 << vs2 << " "
                 << endl;
        }
        //fil << n << " " << d << " " << v << endl;

    }while(d>0.01*dv || it < 100);
    //fil.close();

    // cout << " ned it=" << it << " n=" << n << endl;
    return n;
}

double ner(double phi){
    double fer(double, double);

    int it=0;

    double vs2=2*(phi-phiDL)*QMe;
    double vs0=sqrt(-2*phiDL*QMe);
    double v=sqrt(abs(vs2));
    double dv=0.01*(vs0-v);
    double d=0.0;
    double n=0.0;

    if(v==0.) n=fer(L-v)*dv;
    else{
        n=3.*sqrt(v*dv/2.)*fer(L-v);
        if(dv<1e-12) return n;
    }

    v+=dv;

    if(isnan(n))
        cout << "ner_(1)_gives_nan,_phi=" << phi
             << " _dv=" << dv << endl;

    //ofstream fil("diag/ner.dat");
    //fil << n << " " << d*dv << " " << v << endl;
    while(v<vs0){
        d=2.*v*fer(L-v)/sqrt(v*v - vs2);
        n+=d*dv;
        v+=dv;
        ++it;

        if(isnan(n)){
            cout << "ner_isnan: ";
            cout << fer(L-v) << " "
                 << v*v-vs2 << " "
                 << v*v << " "
                 << vs2 << " "
                 << dv << " "
                 << phi << " "
                 << endl;
        }
        //fil << n << " " << d*dv << " " << v << endl;
    }
    if(isnan(n))
        cout << "ner_(2)_gives_nan,_phi=" << phi
             << " _dv=" << dv << endl;

    n+=v*fer(L-v)*dv/sqrt(v*v-vs2);

    //fil << n << " " << d*dv << " " << v << endl;
    //fil.close();

    if(isnan(n))
        cout << "ner_(3)_gives_nan,_phi="
             << phi << " _dv=" << dv << endl;

    return n;
}

double nia(double phi){
    double fia(double, double);

    int it=0;

    double vs2=2*(phiDL-phi)*Qmp;
```

```
double nia(double phi){
    double fia(double, double);

    int it=0;

    double vs2=2*(phiDL-phi)*Qmp;
```

```

double dv=0.05*vthp;
double n=0.0;
double d=0.0;
double v=0.0;

//ofstream fil("diag/nia.dat");
//fil << n << " " << d << " " << v << endl;
do{
    v+=dv;
    d=v*fia(L,-v)/sqrt(v*v + vs2);
    n+=dv;
    ++it;

    if(isnan(d)){
        cout << "nia_isnan_"
        cout << fia(L,v) << " "
        << v+v*vs2 << " "
        << v*v << " "
        << vs2 << " "
        << endl;
    }

    //fil << n << " " << d << " " << v << endl;

}while(d>0.01*dv || it < 100);
//fil.close();

//    cout << "    nia it=" << it << " n=" << n << endl;
return n;
}

double nid(double phi){
    double fid(double,double);

    int it=0;

    double a=2.*phi*QMp;
    double v=sqrt(2.*phiDL*QMp);
    double dv=0.01*v;
    double d=0.0;
    double n=0.0;

    n=(3./2.)*sqrt(v*dv/2.)*fid(0.,v);
    v+=dv;

    //ofstream fil("diag/nid.dat");
    //fil << n << " " << d << " " << v << endl;
    do{
        d=v*fid(0.,v)/sqrt(v*v - a);
        n+=dv;
        v+=dv;
        ++it;

        if(isnan(d)){
            cout << "nid_isnan_"
            cout << d << " " << n << " "
            << fid(0,v) << " "
            << v*v-a << " "
            << v*v << " "
            << a << " "
            << endl;
        }

        //fil << n << " " << d << " " << v << endl;
    }while(d>0.1*dv || it < 100);
    //fil.close();

    //    cout << "    nid it=" << it << " n=" << n << endl;
    return n;
}

double nir(double phi){
    double fir(double,double);

    int it=0;

    double a=2.*phi*QMp;
    double vs0=sqrt(2.*phiDL*QMp);
    double v=sqrt(2.*(phi)*QMp);
    double dv=0.01*(vs0-v);
    double d=0.0;
    double n=0.0;

    if(v==0.) n=fir(0.,v)*dv;
    else{
        n=3.*sqrt(v*dv/2.)*fir(0.,v);
        if(dv<1e-12) return n;
    }
    v+=dv;

    if(isnan(n))
        cout << "nir_(1)_gives_nan, phi="
        << phi << " dv=" << dv << endl;

    //ofstream fil("diag/nir.dat");
    //fil << n << " " << d*dv << " " << v << endl;
    while(v<vs0){
        d=2.*v*fir(0.,v)/sqrt(v*v - a);
        n+=dv;
        v+=dv;
        ++it;

        if(isnan(n))

```

```

                                cout << "nir_isnan_" << " "
                                << fir(0.,v) << " "
                                << v*v-a << " "
                                << v*v << " "
                                << a << " "
                                << dv << " "
                                << phi << " "
                                << endl;

//fil << n << " " << d*dv << " " << v << endl;
}

if(isnan(n)) cout << "nir_(2)_gives_nan,_phi="
               << phi<< "_dv=" << dv << endl;
n+=v*fir(0.,v)*dv/sqrt(v*v-a);

//fil.close();

if(isnan(n)) cout << "nir_(3)_gives_nan,_phi="
               << phi<< "_dv=" << dv << endl;
// cout << " nir it=" << it << " n="<< n << endl;
return n;
}

```

---

## A.1.7 flux.cpp

---

```

/*      ID Double Layer Simulations
 *      Author : Vegard Lundby Rekaa
 *      University of Oslo, Norway
 *      March, 2009
 */

/*
 *      INITIALIZE FLUX
 *
 */
double NRintegration(double (*f)(double),double l, double u,
int steps){
//Simpsons integration scheme

double dx=(u-l)/(steps-1);
double I=.5*(f)(l)*dx;
for(double x=l; x<u; x+=dx) I += (*f)(x)*dx;
I +=.5*(f)(u)*dx;
return I;
}

void initflux(double alpha[]){
//Finds the expected number of particles to enter at
//each
//timestep. Experiencing a flux in 100 times lower
//than
//what leaves the domain.
//This is independent of what I choose L and dt to be.

flux_ea = alpha[0]*dt*NRintegration(vfea, 0., uea(1x),
200);
flux_erd = alpha[1]*dt*NRintegration(vfer, led(ux),
0., 200);
flux_ia = alpha[2]*dt*NRintegration(vfia, lia(ux),
0., 200);
flux_ird = alpha[3]*dt*NRintegration(vfir, 0., uid(1x),
200);

cout << "_Influx_" << endl;
cout << "Pea="<<flux_ea<<"Perd="<<flux_erd<<"Pia="
<<flux_ia<<"Pird="<<flux_ird<<endl;
}

/*
 *      INFLUX OF PARTICLES
 */
long fluxnum(double P){
//From a double telling the flux, return a long
containing total
//number of particles to be drawn. Can handle non
integer P's.

if(drand48()<=(P-floor(P))) return long(P)+1;
else return long(P);
}

double max(double x1, double x2, double x3, double x4){
//Return the maximum of the 4 double variables.

double m=0.;
if(x1>m)m=x1;
if(x2>m)m=x2;
if(x3>m)m=x3;

```

```

        if(x4>m)m=x4;
        return m;
    }

    void position(long N, double v[], double x[], double x0){
        //From known velocities, find their positions
        double drand;
        for(long i=0; i<N; ++i){
            x[i] = x0 + v[i]*dt*(drand=drand48());
            if(x[i]>ux || x[i]<lx){
                cout
                <<"_flux.cpp:_position():_x_out_of_
                bounds_["<<"x="
                << x[i] << "]" << v[i] << "]" << " " <<
                drand << endl;
            }
        }
    }

    void influx(int time){
        //Generate particles according to their distrobution
        functions
        //vfea, etc. Use the Inversion method supplied in
        Utils.cpp.

        void histogram(int time, string specie, double x[],
            double v[], long Nxv,
            double (*f)(double), double, double);
        long N;

        double m=max(flux_ea, flux_erd, flux_ia, flux_ird);
        x = new double[long(m+1)];
        v = new double[long(m+1)];

        N=fluxnumber(flux_ea);
        Inversion ea(0., uea(lx), 200, vfea, fnorm);
        ea.draw(N,v,false);
        position(N,v,x,ux);
        fill(&Ne,x,ve,N,x,v);

        N=fluxnumber(flux_erd);
        Inversion erd(led(ux), 0., 200, vfer, fnorm);
        erd.draw(N,v,false);
        position(N,v,x,ux);
        fill(&Ne,x,ve,N,x,v);

        N=fluxnumber(flux_ia);
        Inversion ia(lia(ux), 0., 200, vfia, fnorm);
        ia.draw(N,v,false);
        position(N,v,x,ux);
        fill(&Np,xp,vp,N,x,v);

        N=fluxnumber(flux_ird);
        Inversion ird(0., uid(lx), 200, vfir, fnorm);
        ird.draw(N,v,false);
        position(N,v,x,ux);
        fill(&Np,xp,vp,N,x,v);

        delete x,v;

    }

    /*
    *          OUTFLOW OF PARTICLES
    */
    void outflux(int time){
        for(int i=0;i<Np;++i) while(xp[i]>ux||xp[i]<lx){
            --Np;
            xp[i]=xp[Np];
            vp[i]=vp[Np];
        };
        for(int i=0;i<Ne;++i) while(xe[i]>ux||xe[i]<lx){
            --Ne;
            xe[i]=xe[Ne];
            ve[i]=ve[Ne];
        };
    }

    void diagoutflux(int time){
        double f=0.5;
        double vse2=-2.*QMe*phiDL*f;
        double vsp2=2.*QMp*phiDL*f;
        int out_e0a=0, out_e0rd=0, out_eLa=0, out_eLrd=0;
        int out_p0a=0, out_p0rd=0, out_pLa=0, out_pLrd=0;

        //OUT
        for(int i=0;i<Np;++i) while(xp[i]>ux||xp[i]<lx){
            if(xp[i]>ux){
                if(vp[i]*vp[i]>vsp2){
                    out_pLa+=1;
                }else{
                    out_pLrd+=1;
                }
            }else{
                if(vp[i]*vp[i]>vsp2){
                    out_p0a+=1;
                }else{

```

```

                    out_p0rd+=1;
                }
            }
            --Np;
            xp[i]=xp[Np];
            vp[i]=vp[Np];
            tinp[i]=tinp[Np];
            vinp[i]=vinp[Np];
        };
        for(int i=0;i<Ne;++i) while(xe[i]>ux||xe[i]<lx){
            if(xe[i]>ux){
                if(ve[i]*ve[i]>vse2){
                    out_eLa+=1;
                }else{
                    out_eLrd+=1;
                }
            }else{
                if(ve[i]*ve[i]>vsp2){
                    out_e0a+=1;
                }else{
                    out_e0rd+=1;
                }
            }
            --Ne;
            xe[i]=xe[Ne];
            ve[i]=ve[Ne];
            tine[i]=tine[Ne];
            vine[i]=vine[Ne];
        };
    }

    ofstream fout("diag/flux2.dat",ios::app);
    fout << time*dt << " "
    << flux_ea << " "
    << out_e0a << " "
    << out_eLa << " "
    << flux_erd << " "
    << out_e0rd << " "
    << out_eLrd << " "
    << flux_ia << " "
    << out_p0a << " "
    << out_pLa << " "
    << flux_ird << " "
    << out_p0rd << " "
    << out_pLrd << " "
    << " "
    << (double)out_eLa/flux_ea << " "
    << (double)out_eLrd/flux_erd << " "
    << (double)out_p0a/flux_ia << " "
    << (double)out_p0rd/flux_ird << " "
    << endl;
    fout.close();
}

/*
*          FLUX "MAIN" SCHEME
*/
void flux(int time){
    //printTVhist(time);
    int oute=Ne, outp=Np; //Count particles that leave/
    enter domain
    //outflux(time); //Remove particles that left
    the domain
    diagoutflux(time);
    oute--Ne; outp--Np; //Count particles that leave/
    enter domain
    int ine=Ne, inp=Np; //Count particles that leave/
    enter domain
    influx(time); //Insert new particles
    ine=Ne-oute; inp=Np-outp; //Count particles that leave/
    enter domain

    //Write in-/outflux to file
    ofstream fout("diag/flux.dat",ios::app);
    fout << time*dt << " "
    << Ne << " "
    << Np << " "
    << ine << " "
    << oute << " "
    << inp << " "
    << outp << " "
    << -ine+oute-outp+inp << " "
    << endl;
    fout.close();
}

/*
*          1D Double Layer Simulations
*          Author : Vegard Lundby Rckaa
*          University of Oslo, Norway
*          March, 2009
*/

```

## A.1.8 fourier.cpp

```

#endif
#include "fftw.cpp"
#else
#include "FFT.cpp"
#endif
void powerspectrum(double data[], int N){
#ifdef FFTW
void fftw(double [], double [],int,int);

//Converting real data to complex array
double re[N], im[N];
for(int i=0; i<N; ++i){
    re[i]=data[i]; //Real part
    im[i]=0.; //Imaginary part
}
//Fourier transform of complex array
fftw(re,im,N,-1); //-1 : forward transform

//Calculating powerspectrum
for(int i=0; i<N; ++i) data[i]=re[i]*re[i] + im[i]*im[i];
#endif

FFT fft;
double W[N];
fft.forward(W,data,N);
for(int i=0; i<N/2; ++i) data[i]=W[i]*W[i] + W[N-i-1]*W[N-i-1];

//Checking for errors in results
for(int i=0; i<N; ++i) if(isnan(data[i]))
    cout<<"_ps_is_NaN:_"<<i<<endl;
}

int findmodes(double data[], int N, int modes[],
              int modessize, string type){
    static int count=0;
    //Frequency
    int N21=N/2-1;
    double f[N], df=1.;

    for(int i=0; i<=N21; ++i){
        f[i]=i*df;
        f[N-i-1]=(i+1)*df;
    }

    N/=2; //Last half of powerspectrum is
        negative //frequencies. Not interesting!

    int finds=0; //Count no of finds
    double mean=0.0; //Find mean
    double tolerance=8.; //Set tolerance
    for(int i=0; i<N; ++i) mean+=data[i]; mean/=N;
    tolerance*=mean;

    //Finding derivative of powerspectrum
    double der[N-1];
    for(int i=0; i<N-1; ++i) der[i]=data[i+1]-data[i];

    //If local maxima > tolerance >> mean value = peak
    found
    int j;
    int k;
    for(int i=1; i<N-1; ++i)
        if((der[i]<0 && der[i-1]>0) && data[i] >
            tolerance){
            j=0;
            //Run through stored modes
            //If i matches j'th element of modes,
            i is
            //already found, go to next found i.
            //If i doesn't match, j'th element of
            modes is
            //either empty or occupied by another
            mode.
            //If either of these, enter while loop
            .
            while(modes[j] != i && j<modessize){
                //If m[j] is empty, store i in
                m[j]
                if(modes[j] == -1){
                    modes[j]=i;
                    ++finds;
                }
                //If m[j] isn't empty, check m
                [j+1]
                //(by exiting this loop with
                //incrementation of j
                else{
                    ++j;
                }
            }

            k=0;
            if(modes[k]!=-1)
                cout<<"_Found_powerspectrum _
                    modes_"
                    +type+"_";

            while(modes[k]!=-1){
                cout << "m["<<k<<"]="<< f[
                    modes[k]]<<"_";
                ++k;
            }
            cout << endl;
        }
    }

#ifdef MOTHERDIAG
//For diagnostics, print mean and tolerance
char snu[13]; sprintf(snu, "%d", count);
string str="diag/findmodes.";
str=str+type+"."+snu+".dat";
ofstream m(str.c_str());
for(int i=0; i<N; ++i) m<<f[i]<<"_ "<<data[i]<<"_ "
    <<mean<<"_ "<<tolerance<<endl;

m.close();
++count;
#endif
return finds;
}

void windowing(double data[], int N){
    double x, f, dx=10./N;
    int i;

    for(i=0, x=0.; i<N/2; ++i, x+=dx){
        f=tanh(x);
        data[i]*=f;
        data[N-i]*=f;
    }
}

void writemodes(double data[], int N,
               int modes[], double time, string type, double
               dx){
    string tagfile="diag/powertags."+type+".dat";
    string powfile="diag/power."+type+".dat";
    string gnuplotfile="diag/power."+type+".p";
    ofstream tag(tagfile.c_str());
    ofstream pow(powfile.c_str(), ios::app);
    ofstream gnu(gnuplotfile.c_str());

    //If any modes found at all
    if(modes[0]!=-1){
        tag<<"time_"<<time<<"_";
        pow<<time<<"_";
        gnu<<"data=../"<<tagfile<<" "<<endl<<"plot_"<<
    }

    double power, wavenumber;
    int j=0;
    while(modes[j]!=-1){
        power=data[modes[j]];
        // wavenumber=f[modes[j]];
        wavenumber=2.*pi*modes[j]/(dx*N);
        tag<< setprecision(4) <<wavenumber<<"_";
        pow<<power<<"_";
        gnu <<"_data_using_1:"<<j+2
            <<"_title_"<<j+2<<"_with_lines_";

        if(modes[j+1]!=-1)gnu<<" ";
        else gnu<<endl;
        ++j;
    }
    pow<<endl;
}

void finalizemother(){
    ofstream left("diag/powertags.left.dat", ios::app);
    ofstream right("diag/powertags.right.dat", ios::app);

    left<<endl; right<<endl;
    left.close(); right.close();

    string cmd1="cat diag/power.left.dat>>_diag/powertags
        .left.dat";
    string cmd2="cat diag/power.right.dat>>_diag/
        powertags.right.dat";
    system(cmd1.c_str());
    system(cmd2.c_str());
}

void mother(double rho[], int N, int time){
    //Split rho into left of- and right of DL
    //Assuming equal amount of space on each side of DL
    double dx=(ux-lx)/(N-1);
    int size=int(-lx/dx); //Size of added plasma in
        array inidces
    int M=N; while(M>size) M/=2; //Finds biggest M=2^n<
        size

```



```

int j=N-1-M; //Muched used index

if(M>2048){
    cout<<"fourier.cpp: _mother() : _Size , _M_too_big"
    <<endl;
    cout<<"Forcing_exit"<<endl;
    exit(1);
}

static double rps[2048]; //Arrays for
    accumulation power-
static double lps[2048]; // spectrum over time
double left[M], right[M];
for(int i=0; i<M; ++i){
    left[i]=rho[i]; //Temporary arrays
    right[i]=rho[i+j];
}

//Find array index of modes
const int modessize=50; //Max modes to be
    found
static int lmodes[modessize]; //Storing of mode
    wavenumbers
static int rmodes[modessize]; //Storing of mode
    wavenumbers
static int found; //Count no. of found
    modes
static int w=1; //Count no. write to
    datafile
static bool initial=true; //When first time run,
    do initial

if(initial){
    for(int i=0; i<modessize; ++i){
        rmodes[i]=-1;
        lmodes[i]=-1;
    }
    for(int i=0; i<M; ++i){
        rps[i]=0.0;
        lps[i]=0.0;
    }
    initial=false;
}

//Diagnostics: taking copies of left, right for
    plottable datafile
int k;
double Lsignal[M], Rsignal[M]; //Signal
double Lwindow[M], Rwindow[M]; //Signal with window
//double Lpowers[M], Rpowers[M]; //
    Powerspectrum
//Saving original signal
for(k=0; k<M; ++k){ Lsignal[k]=left[k]; Rsignal[k]=right
    [k];}

//Windowing
windowing(left, M);
windowing(right, M);
for(k=0; k<M; ++k){ Lwindow[k]=left[k]; Rwindow[k]=right
    [k];}
//Powerspectrum
powerspectrum(left, M);
powerspectrum(right, M);
//for(k=0; k<M; ++k) Lpowers[i]=left[k], Rpowers=right[k
    ];

//Accumulating over time
for(int i=0; i<M; ++i){
    rps[i] += right[i];
    lps[i] += left[i];
}

if(10*time/N_time == w){
#ifdef MOTHERDIAG
    //Diagnostics
    char snu[13]; sprintf(snu, "%d", time);
    string file="diag/mother.signal.";
    file=file+snu+".dat";
    ofstream fout(file.c_str());

    double l, r;
    for(k=0, l=lx, r=lx+j*dx ; k<M ; ++k, l+=dx, r+=dx
    ){
        cout << l << " " //1. Left
        positions
        << r << " " //2. Right
        positions
        << Lsignal[k] << " "
        << Rsignal[k] << " "
        << Lwindow[k] << " "
        << Rwindow[k] << " "
        <<endl;
    }
    fout.close();
#endif

    //Search for new modes
    found+=findmodes(rps, M, rmodes, modessize, "
        right");
    found+=findmodes(lps, M, lmodes, modessize, "left
        ");
    //Write power of modes to file

    writemodes(rps, M, rmodes, time*dt, "right", dx);
    writemodes(lps, M, lmodes, time*dt, "left", dx);
    //Reset accumulated arrays
    for(int i=0; i<M; ++i){
        rps[i]=0.0;
        lps[i]=0.0;
    }

    ++w;
}

}

void testmother(){
    int N=1024;
    double RHO[N];

    double a=1, b=1, c=.0, d=0.000;
    void mother(double *, int, int);
    double dx=8.*pi/N;
    for(int k=0; k<40; ++k){
        for(int i=0; i<N; ++i)
            RHO[i]= (a+d*drand48())*sin(i*dx)
            // RHO[i]= sin(3.*i*dx)+cos(7.*i*dx)
            ;
            +(b+d*drand48())*cos(10*i*dx)
            +(c+d*drand48())*cos(34*i*dx)
            +0.5*(2.*drand48()-1.);
        mother(RHO, N, k);
        d=0.0002;
    }

    cout << "Fourier(mother)_routines_test_finished , _
        exiting."<<endl;
    exit(0);
}

void FFTphidl(double data[], int N, double delta){
    void windowing(double*, int);
    void powerspectrum(double*, int);

    //Finding highest M=2^p < N
    int M=int(pow(2., 20.));
    for(double p=20.; M<N; --p) M=int(pow(2., p));

    //Declaring data array of M
    double f[M]; int i, j;
    //Copying last M elements of data
    for(i=0, j=N-M; i<M; ++i, ++j) f[i]=data[j];

    //Powerspectrum with window
    windowing(f, M);
    powerspectrum(f, M);

    //Write results to file
    ofstream fout("diag/phidl.powerspectrum.dat");
    for(i=0; i<M; ++i) fout<<2.*pi*i/(delta*M)<<" "<<
        f[i]<<endl;
    fout.close();
}

}

/* 1D Double Layer Simulations
 * Author : Vegard Lundby Rekaa
 * University of Oslo, Norway
 * March, 2009
 */

void finalize(bool);

int diagcounter=0;
int carpetcounter=0;

void initialdiagnostics(double PHI[], int N_time, int N){
    void densityplot(double [], int, int);
    void initTVhist(long, long);
    void resetTVhist();
    void gnuplotpartnum();

    densityplot(PHI, N, 2);
    initTVhist(Ne, Np);
    resetTVhist();
    gnuplotpartnum();

    void printf(double (*fa)(double, double), double (*la)
        (double),

```

## A.1.9 diagnostics.cpp

```

        double (*ua)(double), double (*fr)(double,
        double),
        double (*lr)(double), double (*ur)(double),
        double (*fd)(double, double), double (*ld)(
        double),
        double (*ud)(double), string specie);

    cout<<endl<<"_Average_no._of_particles_pr._cell=_"<<
        double(Ne*Np)/(2.*N)<<endl<<endl;
    cout<<"Printf_takes_time";
    printf(fea, lea, uea, fer, ler, uer, fed, led, ued, "e
    ");
    printf(fia, lia, uia, fir, lir, uir, fid, lid, uid, "i
    ");
    cout<<"_done."<<endl;

    ofstream phiout("diag/initialphi.dat");
    double x,dx=(ux-lx)/(N-1);
    for(x=lx; x<=ux; x+=dx) phiout<<x<<"_"<<phi(x)<<endl;
    phiout.close();
}

void gnuplotpartnum(){
    ofstream fout("diag/partnum.dat");
    fout<<"_Ne"<<Ne<<"_Ni"<<Np<<endl;
}

void diagnostics(double RHO[], double PHI[], double E[], int N
, int time){

    void checkforan(double [], double [], double [], int)
    ;
    void phidprintout(int,double);
    void fields(double [], double [], double [], int, int)
    ;
    void densityplot(double [], int, int);
    void ps_plot(string, double [], double, double,
        double [], double, double, long, int);
    void boundaryvdf(double x[], double v[], long Nxv,
        double (*fa)(double, double), double la,
        double ua,
        double (*frd)(double, double), double lrd,
        double urd,
        int, string);
    void fieldcarpet(int,double [], double [], double [],
        int);
    void gnuplottimestamps(double);
    void gnuplotmark(string,string);

    void mother(double*, int, int);
    mother(E,N,time);

    phidprintout(time, (PHI[N-1]-PHI[0]) );

    //Data written 200 times during simulations
    if(time*100/N_time==carpetcounter){
        fieldcarpet(time,RHO,PHI,E,N);
        ++carpetcounter;
    }

    //Data written 10 times during simulations
    //plotting routines based on the fact that the number
    10 is 10!
    if(time*10/N_time==diagcounter){
        cout << "_Diagnostics_at_t="<<time;

        gnuplottimestamps(time*dt);

        //print_influzhist(0.,uid(lx),time,"ir");
        checkforan(RHO,PHI,E,N); //Check ALL value for
        NaN
        fields(RHO, PHI, E, N, time);
        densityplot(PHI, N, time);
        ps_plot("xe-ve",xe,lx,ux,ve,led(ux),uea(ux),Ne
        ,time);
        ps_plot("xp-vp",xp,lx,ux,vp,lia(lx),uid(lx),Np
        ,time);

        boundaryvdf(xe,ve,Ne,fea,lea(0.),
        uea(0.),fer,led(L),uer(L),time,"e");
        boundaryvdf(xp,vp,Np,fia,lia(L),
        uia(L),fir,lir(0.),uid(0.),time,"p");

        diagcounter++;
        cout << "_... Finished"<<endl;
    }
}

void gnuplottimestamps(double time){

    char snu[13]; sprintf(snu, "%d", diagcounter);
    string str="gnuplot/marks/time";
    str = str+snu;
    ofstream fout(str.c_str());

    fout.precision(3);
    fout<<time<<endl;
    fout.close();
}

void gnuplotmark(string name, string file){

    char snu[13]; sprintf(snu, "%d", diagcounter);
    string str="gnuplot/marks/";
    str = str+name+snu;
    ofstream fout(str.c_str());
    fout<<"./"<<file;
    fout.close();
}

void fieldcarpet(int time, double RHO[], double PHI[],
        double E[], int N){
    ofstream carpet("diag/carpet.dat", ios::app);

    double x, dx=(ux-lx)/(N-1);
    int i;
    for(i=0, x=lx; i<N; ++i,x+=dx)
        carpet<<time*dt<<"_"<<x<<"_"
            <<RHO[i]<<"_"<<PHI[i]<<"_"<<E[i]<<endl
            ;

    carpet<<endl;
    carpet.close();
}

void phidprintout(int time, double dl){
    //phiDL printout
    ofstream pout("diag/phidl.dat",ios::app);
    pout<<time*dt<<"_"<<dl<<endl;
    pout.close();
    if(phiDL<0){
        cout<<endl<<"_ERROR:_phiDL<0"<<endl;
        finalize(true);
    }
}

void fields(double RHO[], double PHI[], double E[], int N, int
time){

    char snu[13]; sprintf(snu, "%d", time);
    string str="diag/fields.";
    str = str+snu+".dat";
    ofstream fout(str.c_str());

    gnuplotmark("fields",str);

    double dx=(ux-lx)/(N-1);
    double x;
    for(int i=0; i<N; ++i)
        fout << i*dx+lx << "_"
            << RHO[i] << "_"
            << PHI[i] << "_"
            << E[i] << "_"
            << endl;

    fout.close();
}

void ps_plot(string specie, double x[], double xl, double xu,
        double v[], double vl, double vu, long N, int time){

    int xbins=200, vbins=xbins;
    double dx=(xu-xl)/(xbins-1), dv=(vu-vl)/(vbins-1);
    double hist[xbins][vbins];
    int ix,iv;
    double X,V;
    //Zeroing histogram
    for(ix=0; ix<xbins; ++ix) for(iv=0;iv<vbins; ++iv) hist
        [ix][iv]=0.;
    //Filling histogram
    for(long i=0;i<N++i)
        ++hist[ int((x[i]-xl)/dx) ][ int((v[i]-vl)/dv)
        ];

    char snu[13]; sprintf(snu, "%d", time);
    string str="diag/ps.";
    str = str+specie+"."+snu+".dat";
    ofstream ps(str.c_str());

    gnuplotmark("ps"+specie, str);

    for(ix=0, X=xl; ix<xbins; ++ix, X=ix*dx+xl){
        for(iv=0, V=vl; iv<vbins; ++iv, V=iv*dv+vl){
            ps << X << "_" << V << "_"
                << hist[ix][iv] << endl;
        }
    }
}

```

```

        ps << endl;
    }

double min(double a[], long N){
    double m=le300;
    for(long i=0; i<N; ++i) if(a[i]<m) m=a[i];
    return m;
}
double max(double a[], long N){
    double m=-le300;
    for(long i=0; i<N; ++i) if(a[i]>m) m=a[i];
    return m;
}

void boundaryvdf(double x[], double v[], long Nxv,
    double (*fa)(double, double), double la, double ua,
    double (*frd)(double, double), double lrd, double urd,
    int time, string specie){
    //VDF AT BOUNDARIES

    double vl=min(v,Nxv);
    double vu=max(v,Nxv);
    //vl=-0.5;
    //vu=0.5;
    //cout << "Histogram vdf at boundary: "<< specie <<
        endl;
    //cout << "vl and vu: "<< vl << " " << vu << endl;

    int bins=100;
    double bin=(vu-vl)/(bins-1);
    long vhu[bins]; //Velocity histogram lower
    long vhl[bins]; //Velocity histogram upper
    for(int i=0; i<bins; ++i){
        vhl[i]=0;
        vhu[i]=0;
    }

    double f=0.002;
    int iv;
    for(long i=0; i<Nxv; ++i){
        iv=int((v[i]-vl)/(vu-vl)*(bins-1));
        if(iv>=0 && iv<bins){
            if(x[i]<lx+f*L){
                if(x[i]<lx) cout << "vdfplot: x<0:_"<<
                    x[i]<<"_"<<v[i]<<endl;
                vhl[iv]+=1;
            }else if(x[i]>ux-f*L){
                if(x[i]>ux) cout << "vdfplot: x>L:_"<<
                    x[i]<<"_"<<v[i]<<endl;
                vhu[iv]+=1;
            }
        }
    }

    //cout << "vhl=";
    //for(int i=0; i<bins; ++i) cout << vhl[i] << " ";cout
        << endl;
    //cout << "vhu=";
    //for(int i=0; i<bins; ++i) cout << vhu[i] << " ";cout
        << endl;

    const int t=time;
    char snu[13]; sprintf(snu, "%d", t);
    string str="diag/vh";
    str=str+specie+"."+snu+".dat";

    gnuplotmark("vdf"+specie, str);

    double V;
    ofstream fvh(str.c_str(), ios::app);
    for(int i=0; i<bins; ++i) {
        V=(vu-vl)*(double(i)+0.5)/double(bins-1)+vl;
        fvh << V << "_"
            << vhl[i] << "_"
            << (*fa)(0.,V) << "_"
            << (*frd)(0.,V) << "_"
            << vhu[i] << "_"
            << (*fa)(L,V) << "_"
            << (*frd)(L,V) << "_"
            << endl;
    }
    fvh.close();

double CIC(double x, double l, double u, double array[], int N
){
    //Returns a linear interpolated value from array[],
    //given at a
    //position(etc.) x in domain [l,u] (lower, upper).
    //Size of array is N

    double w=(x-l)/(u-l)*(N-1);
    int m=int(floor(w));
    w-=m;

    if(w>1. || w<0.){
        cout<<"_CIC():_w_out_of_range:_"<<w<<endl;
        cout<<"_"<<m<<"_"<<l<<"_"<<u<<"_"<<x<<"_"
            <<endl;
    }
    if(m<0 || m>=N-1){
        cout<<"_CIC():_m_out_of_range:_"<<m<<endl;
        cout<<"_"<<m<<"_"<<l<<"_"<<u<<"_"<<x<<"_"
            <<endl;
    }

    return (1.-w)*array[m] + w*array[m+1];
}

void densityplot(double PHI[], int N, int time){
    int m;
    int bins=200;
    double bin=(ux-lx)/(bins-1);
    long ea[bins];
    long er[bins];
    long ia[bins];
    long ir[bins];

    for(int i=0; i<bins; ++i){
        ea[i]=0;
        er[i]=0;
        ia[i]=0;
        ir[i]=0;
    }

    double x, vv, vs2;
    double f=0.50;
    for(long i=0; i<Ne; ++i){
        x=xe[i];
        vv=ve[i]*ve[i];
        vs2=2.*Qm*f*(phiDL-CIC(x,lx,ux,PHI,N));
        if(vv<vs2 && x>L/2.) er[int((x-lx)/bin)
            ]+=1;
        else ea[int((x-lx)/bin)
            ]+=1;
    }
    for(long i=0; i<Np; ++i){
        x=xp[i];
        vv=vp[i]*vp[i];
        vs2=2.*Qm*f*(phiDL-CIC(x,lx,ux,PHI,N));
        if(vv<vs2 && x<L/2.) ir[int((x-lx)/bin)
            ]+=1;
        else ia[int((x-lx)/bin)
            ]+=1;
    }

    char snu[13]; sprintf(snu, "%d", time);
    string str="diag/density.";
    str=str+snu+".dat";
    gnuplotmark("density", str);

    ofstream fout(str.c_str());
    for(int i=0; i<bins-1; ++i)
        fout <<i*bin+lx<<"_"
            <<ea[i]/bin<<"_"
            <<er[i]/bin<<"_"
            <<ia[i]/bin<<"_"
            <<ir[i]/bin<<"_"
            <<ea[i]/bin+er[i]/bin<<"_"
            <<ia[i]/bin+ir[i]/bin<<"_"
            <<endl;
    fout.close();

bool checknan(string type, double x, int i){
    if(isnan(x)){
        cout << type <<"_isnan:_"<< i << "_"<<x<<endl
            ;
        finalize(true);
        return true;
    }else return false;
    }

void checkfornan(double RHO[], double PHI[], double E[], int N
){
    cout << "_Checking_for_nans_"
        ;
    bool stop=false;
    for(int i=0; i<N; ++i){
        stop=checknan("rho",RHO[i],i);
        stop=checknan("phi",PHI[i],i);
        stop=checknan("E",E[i],i);
    }
    for(long i=0; i<Ne; ++i){
        if(isnan(xe[i])){
            cout<<"xe:_"<<i<<"_"<<xe[i]<<endl;
            break;
        }
        if(isnan(ve[i])){
            cout<<"ve:_"<<i<<"_"<<ve[i]<<endl;
            break;
        }
        if(isnan(dve[i])){
            cout<<"dve:_"<<i<<"_"<<dve[i]<<endl;
            break;
        }
    }
    for(long i=0; i<Np; ++i){

```

```

        if(isnan(xp[i])){
            cout<<"xp_:"<<i<<"_ "<<xp[i]<< endl;
            break;}
        if(isnan(vp[i])){
            cout<<"vp_:"<<i<<"_ "<<vp[i] << endl;
            break;}
        if(isnan(dvp[i])){
            cout<<"dvp_:"<<i<<"_ "<<dvp[i]<<endl;
            break;}
    }

    if(stop){
        cout << "Forcing_exit_due_to_NaN's"<<endl;
        finalize(true);
    }
}

void print_simdetails(double vtdx, int &N){
    //Stores total simulationtime as a system variable
    //char snu[13]; sprintf(snu, "%d", Ntime);
    //string str="export CRUNSIMTIME=";
    //str=str+snu;
    //system(str.c_str());

    double mach=abs(drifte/vthe);
    double dx=(ux-lx)/(N-1);
    ofstream params("diag/params.dat");
    params <<"_l_="<<mach<<"_p="<<phiDL<<"_m="<<mp
    /me<<"_t="<<N_time<<"_dt="<<dt;
    params << setprecision(3) << N_time*dt << " {/Symbol_w
    }_pe}_{-1}";
    params <<"_Nc="<<N<<"_l_="<<ux;
    params << setprecision(2) <<"_dt="<< dt <<"_dx="
    << dx
    <<"_vdt/dx="<< vtdx
    <<"_l_="<<ux;

    cout <<endl;
    cout <<"_Simulation_parameters:_ " <<endl
    <<"_l_="<<ux<<"_p="<<mach<<"_m="<<mp
    <<"_t="<<N_time<<"_dt="<<dt;
    //<<"_L = "<<L <<endl
    //<<"_lx = "<<lx <<endl
    //<<"_ux = "<<ux <<endl
    <<"_dx="<<dx <<endl
    <<"_dt="<<dt <<endl
    <<"_vdt/dx="<<vtdx <<endl
    <<"_time="<<N_time <<endl
    <<"_time*dt="<<N_time*dt <<endl
    <<endl;

    cout <<"_Plasma_and_DL_parameters:_ " <<endl
    <<"_mi/me="<<mp/me <<endl
    <<"_Ti/Te="<<vthp*vthp/me*mp<<endl
    <<"_mach="<<mach <<endl
    <<"_phiDL="<<phiDL <<endl
    <<endl;

#ifdef BC_vND
    cout<<"_Boundary_condition:_vN-_D"<<endl;
    params<<"_vN-D";
#endif
#ifdef BC_DD
    cout<<"_Boundary_condition:_D-_D"<<endl;
    params<<"_D-D";
#endif
#ifdef BC_vNvN
    cout<<"_Boundary_condition:_vN-_vN"<<endl;
    params<<"_vN-vN";
#endif
#ifdef BC_DD
    #ifndef BC_vND
    #ifndef BC_vNvN
    cout<<"_Boundary_condition:_D-_vN"<<endl;
    params<<"_D-vN";
    #endif
    #endif
    #endif
    params.close();
#ifdef PTEST
    cout<<"_Running_test_on_Poisson_solver..._"<<endl;
#endif
}

void initialstabilitycheck(double vtdx, double mach, double
TionTe, int &N){
    bool stop=false;

    cout << "_Initial_stabilitycheck..._";

    //Test--something--something
    if(vtdx >= 0.6) cout << " dt too large! ", stop=true;

    //Bohm criterium
    if(mach<=(3.0+TionTe))
        cout<<"_Bohm_criterium_violated!_"<<endl;
    else
        cout<<"_Bohm_criterium_OK!"<<endl;

    //
}

//Strong double layer
if(phiDL<10.)
    cout << "_Weak_double_layer!_"<<endl;

//f(v=upper limit / lower limit)=0
double zero=1e-10;
double ea0=fea(0.,uea(0.));
double eaL=fea(L,uea(L));
double ed0=fed(0.,led(0.));
double edL=fed(L,led(L));
double ia0=fia(0.,lia(0.));
double iaL=fia(L,lia(L));
double id0=fid(0.,uid(0.));
double idL=fid(L,uid(L));
if(ea0>zero) cout<<"_fea(0,inf)="<<ea0<<"_<<endl;
if(eaL>zero) cout<<"_fea(L,inf)="<<eaL<<"_<<endl;
if(ed0>zero) cout<<"_fed(0,-inf)="<<ed0<<"_<<endl;
if(edL>zero) cout<<"_fed(L,-inf)="<<edL<<"_<<endl;
if(ia0>zero) cout<<"_fia(0,-inf)="<<ia0<<"_<<endl;
if(iaL>zero) cout<<"_fia(L,-inf)="<<iaL<<"_<<endl;
if(id0>zero) cout<<"_fid(0,inf)="<<id0<<"_<<endl;
if(idL>zero) cout<<"_fid(L,inf)="<<idL<<"_<<endl;

//If important stabilitytest violated, stop
simulations
if(stop){
    cout <<"_forcing_exit!"<<endl<<endl;
    exit(1);
}else{
    cout << "_Finished" << endl;
}

}

void stabilitycheck(double phi[], double rho[], double E[],
int N){
    cout << "_Stabilitycheck:";
    //phi(x)>0
    for(int i=0; i<N; ++i)
        if(phi[i]<0.){cout<<"_phi(x)<0"; break;}

    //E<0
    for(int i=0; i<N; ++i)
        if(E[i]>0.){ cout<<"_E(x)>0"; break;}

    //rho
    double rhosum=0., dx=(ux-lx)/(N-1), m=0.0;
    for(int i=0; i<N; ++i){
        rhosum+=rho[i]; //Total charge
        if(m>abs(rho[i])) m=abs(rho[i]); //Maximum
        value
    }
    double total=rhosum*dx/m*100.; //Total rho to
    max in %
    double lb=abs(rho[0])*dx/m*100.; //Lower
    boundary
    double ub=abs(rho[N-1])*dx/m*100.; //Upper
    boundary
    double acceptance=5.; //Unit
    percentage
    if(lb>acceptance) cout << "_rho(0)>>0";
    if(ub>acceptance) cout << "_rho(L)>>0";
    if(total>acceptance) cout << "_sum(rho)>>0";

    cout << "..._Finished" << endl;
}

void print(double (*fa)(double, double), double (*la)(
double),
double (*ua)(double), double (*fr)(double,
double),
double (*lr)(double), double (*ur)(double),
double (*fd)(double, double), double (*ld)(
double),
double (*ud)(double), string specie){

    int Nx=200, Nv=200;

    double dx=(ux-lx)/(Nx-1);
    double vmax=0.0, vmin=0.0, tmp;
    for(double x=lx; x<ux; x+=dx){
        tmp=(*ua)(x); if(vmax < tmp) vmax=tmp;
        tmp=(*ud)(x); if(vmax < tmp) vmax=tmp;
        tmp=(*la)(x); if(vmin > tmp) vmin=tmp;
        tmp=(*ld)(x); if(vmin > tmp) vmin=tmp;
    }
    double dv=(vmax-vmin)/(Nv-1);

    double x,v,f;

```

```

string str="diag/printf.";
str=str+specie+"_dat";
ofstream fout(str.c_str());

for(double x=lx; x<=ux; x+=dx){
    for(double v=vmin; v<=vmax; v+=dv){
        if(((*la)(x)<v && v<(*ua)(x))
            f=(*fa)(x,v);
        else if(((*lr)(x)<=v && v<=(*ur)(x))
            f=(*fr)(x,v);
        else if(((*ld)(x)<v && v<(*ud)(x))
            f=(*fd)(x,v);
        else
            f=0.0;

        fout<<f<<"_ " <<v<<"_ " <<x<<endl;
    }
    fout << endl;
}
fout.close();
}

```

## A.2 Gnuplot scripts

### A.2.1 Output to .ps script

```

load 'script.header.p'

plotfields      =1
plotcarpet      =0
plotdensity     =1
plotflux        =1
plotvdf         =1
plotps         =1
plotpower       =1

set terminal postscript enhanced color font "Times-Roman,8"
set output "../plot/results.ps"

tittel="cat_../diag/partnum.dat ' ' cat_../diag/params.dat '\n' '
cat_../diag/date.dat ' '
carpettittel="cat_../diag/partnum.dat '\n' cat_../diag/params.
dat ' "

if(plotfields) load 'results.fields.p'
if(plotcarpet) load 'results.carpet.p'
if(plotdensity) load 'results.density.p'
if(plotflux) load 'results.flux.p'
if(plotvdf) load 'results.vdf.p'
if(plotps) load 'results.phasespace.p'
if(plotpower) load 'results.power.p'

set out

load 'script.footer.p'

```

### A.2.2 Output to multiple .eps files script

```

load 'script.header.p'

plotfields      =1
plotcarpet      =1
plotdensity     =1
plotflux        =1
plotvdf         =1
plotps         =1
plotpower       =1

set terminal postscript enhanced eps color font "Times-Roman,
,11"

tittel="cat_../diag/partnum.dat ' ' cat_../diag/params.dat '\n'
carpettittel="cat_../diag/partnum.dat '\n' cat_../diag/params.
dat ' "

if(plotfields) set output "../plot/results.fields.eps"
if(plotfields) load 'results.fields.p'
if(plotfields) set out

if(plotcarpet) set output "../plot/results.carpet.eps"
if(plotcarpet) load 'results.carpet.p'
if(plotcarpet) set out

```

```

if(plotdensity) set output "../plot/results.density.eps"
if(plotdensity) load 'results.density.p'
if(plotdensity) set out

if(plotflux) set output "../plot/results.flux.eps"
if(plotflux) load 'results.flux.p'
if(plotflux) set out

if(plotvdf) set output "../plot/results.vdf.eps"
if(plotvdf) load 'results.vdf.p'
if(plotvdf) set out

if(plotps) set output "../plot/results.phasespace.eps"
if(plotps) load 'results.phasespace.p'
if(plotps) set out

if(plotpower) set output "../plot/results.power.eps"
if(plotpower) load 'results.power.p'
if(plotpower) set out

set out

load 'script.footer.p'

```

### A.2.3 script.header.p

```

# GNUPLOT file for plotting results based on fields from
# Double Layer
# simulations as a part of my masters thesis. -Vegard L. Rekaa
# Based on GNUPLOT 4.2

set macros # enable macros
set autoscale # scale axes
set autoscale automatically
unset log # remove any log-
set scaling
unset label # remove any previous
set labels
unset title
unset xlabel
unset ylabel
set xtic auto # set xtics
set xtic automatically
set ytic auto # set ytics
set ytic automatically

```

### A.2.4 script.footer.p

```

set out
set term x11
unset title
unset label
unset xlabel
unset ylabel
unset size
set xtics auto
set ytics auto
set autoscale

```

### A.2.5 results.fields.p

```

time0=" ' ' cat_marks/time0 ' ' "
time1=" ' ' cat_marks/time1 ' ' "
time2=" ' ' cat_marks/time2 ' ' "
time3=" ' ' cat_marks/time3 ' ' "
time4=" ' ' cat_marks/time4 ' ' "
time5=" ' ' cat_marks/time5 ' ' "
time6=" ' ' cat_marks/time6 ' ' "
time7=" ' ' cat_marks/time7 ' ' "
time8=" ' ' cat_marks/time8 ' ' "
time9=" ' ' cat_marks/time9 ' ' "
time10=" ' ' cat_marks/time10 ' ' "

fields0=" ' ' cat_marks/fields0 ' ' "
fields1=" ' ' cat_marks/fields1 ' ' "
fields2=" ' ' cat_marks/fields2 ' ' "
fields3=" ' ' cat_marks/fields3 ' ' "
fields4=" ' ' cat_marks/fields4 ' ' "
fields5=" ' ' cat_marks/fields5 ' ' "
fields6=" ' ' cat_marks/fields6 ' ' "
fields7=" ' ' cat_marks/fields7 ' ' "
fields8=" ' ' cat_marks/fields8 ' ' "
fields9=" ' ' cat_marks/fields9 ' ' "
fields10=" ' ' cat_marks/fields10 ' ' "

```

```

set xlabel 'x_{/Symbol_1}_D]'

set multiplot layout 2,2 title tittel

set key off
options="using_1:2_w_l_t"
set title "Charge_density"
set ylabel '{/Symbol_r}(x)_{en_0}'
plot fields0 @options @time0, fields1 @options @time1, fields2
    @options @time2, fields3 @options @time3, fields4
    @options @time4, fields5 @options @time5, fields6
    @options @time6, fields7 @options @time7, fields8
    @options @time8, fields9 @options @time9, fields10
    @options @time10

set key box samplen 2 width 0 title 't_{/Symbol_w}_{pe}^{-1}]'
    ' outside left center
options="using_1:3_w_l_t"
set title "Potential"
set ylabel '{/Symbol_f}(x)_{/Symbol_k}T/e]'
plot fields0 @options @time0, fields1 @options @time1,
    fields2 @options @time2, fields3 @options @time3,
    fields4 @options @time4, fields5 @options @time5,
    fields6 @options @time6, fields7 @options @time7,
    fields8 @options @time8, fields9 @options @time9,
    fields10 @options @time10

set key off
options="using_1:4_w_l_t"
set title "Electric_field"
set ylabel 'E(x)_{/Symbol_k}T/e{/Symbol_1}_D]'
plot fields0 @options @time0, fields1 @options @time1,
    fields2 @options @time2, fields3 @options @time3,
    fields4 @options @time4, fields5 @options @time5,
    fields6 @options @time6, fields7 @options @time7,
    fields8 @options @time8, fields9 @options @time9,
    fields10 @options @time10

set key off
set title "Evolution_of_{/Symbol_f}_{DL}_in_time"
set xlabel "t_{/Symbol_w}_{pe}^{-1}]"
set ylabel '{/Symbol_f}_{DL}(t)_{/Symbol_k}T/e]'
plot './diag/phiDL.dat' w l t "phiDL(time)"

set key default
unset multiplot

```

## A.2.6 results.carpet.p

```

set view 60,60
set pm3d at bs
set autoscale
set xlabel 't_{/Symbol_w}_{pe}^{-1}]'
set ylabel 'x_{/Symbol_1}_D]'
set key off

set title tittel
set ylabel '{/Symbol_r}(x,t)_{en_0}'
splot './diag/carpet.dat' using 1:2:3 with pm3d

#set title "Potential\n_'cat_../diag/params.dat"
#set ylabel "{/Symbol_f}(x,t)_{/Symbol_k}T/e]"
#splot './diag/carpet.dat' using 1:2:4 with pm3d

#set title "Electric_field\n_'cat_../diag/params.dat"
#set ylabel "E(x,t)_{/Symbol_k}T/e{/Symbol_1}_D]"
#splot './diag/carpet.dat' using 1:2:5 with pm3d

unset xlabel
unset ylabel
unset zlabel
unset title
unset pm3d

```

## A.2.7 results.density.p

```

p0="cat_marks/density0"
p1="cat_marks/density2"
title1="t=cat_marks/time0'_{vs,t=cat_marks/time2'_{/Symbol_w}
    }_{pe}^{-1}]"
p2="cat_marks/density4"
title2="t=cat_marks/time0'_{vs,t=cat_marks/time4'_{/Symbol_w}
    }_{pe}^{-1}]"
p3="cat_marks/density6"
title3="t=cat_marks/time0'_{vs,t=cat_marks/time6'_{/Symbol_w}
    }_{pe}^{-1}]"

```

```

plot0="p0_u_1:2_w_l_t_'ea',_p0_u_1:3_w_l_t_'er',_p0_u_1:4_w_l_
    t_'ia',_p0_u_1:5_w_l_t_'ir'"
plot1="p1_u_1:2_w_l_t_'ea',_p1_u_1:3_w_l_t_'er',_p1_u_1:4_w_l_
    t_'ia',_p1_u_1:5_w_l_t_'ir'"
plot2="p2_u_1:2_w_l_t_'ea',_p2_u_1:3_w_l_t_'er',_p2_u_1:4_w_l_
    t_'ia',_p2_u_1:5_w_l_t_'ir'"
plot3="p3_u_1:2_w_l_t_'ea',_p3_u_1:3_w_l_t_'er',_p3_u_1:4_w_l_
    t_'ia',_p3_u_1:5_w_l_t_'ir'"
total0="p0_u_1:6_w_l_t_'e',_p0_u_1:7_w_l_t_'i'"
total1="p1_u_1:6_w_l_t_'e',_p1_u_1:7_w_l_t_'i'"
total2="p2_u_1:6_w_l_t_'e',_p2_u_1:7_w_l_t_'i'"
total3="p3_u_1:6_w_l_t_'e',_p3_u_1:7_w_l_t_'i'"

set autoscale
set xlabel 'x_{/Symbol_1}_D]'
set multiplot layout 2,3 columnsfirst title tittel

set key outside left center box title 'Specie' samplen 2
set title title1
#set yrange[-100:5500]
set ylabel 'n_{sp}(x)_{n_0}'
plot @plot0, @plot1
set ylabel 'n_{s}(x)_{n_0}'
#set yrange [1000:5500]
plot @total0, @total1

set key off
set title title2
#set yrange[-100:5500]
set ylabel 'n_{sp}(x)_{n_0}'
plot @plot0, @plot2
#set yrange [1000:5500]
set ylabel 'n_{s}(x)_{n_0}'
plot @total0, @total2

set title title3
#set yrange[-100:5500]
set ylabel 'n_{sp}(x)_{n_0}'
plot @plot0, @plot3
#set yrange [1000:5500]
set ylabel 'n_{s}(x)_{n_0}'
plot @total0, @total3

unset multiplot
set autoscale

```

## A.2.8 results.flux.p

```

doplot=0
if(doplot) set multiplot layout 4,1 title "Flux_\n_'cat_
    ../diag/partnum.dat'_{cat_../diag/params.dat'_{n_'cat_
    ../diag/date.dat'"

if(doplot) set xrange ['cat_marks/time0'_{cat_marks/
    time10']
if(doplot) flux="../diag/flux.dat"
if(doplot) set xlabel "Time_{/Symbol_w}_{pe}^{-1}]"
if(doplot) set key bottom left box
if(doplot) set title "Number_of_particles"
if(doplot) plot flux u 1:2 w l t "Electrons", flux u
    1:3 w l t "Ions"
if(doplot) set key top left box
if(doplot) set title "Total_flux,_electrons"
if(doplot) plot flux u 1:4 w l t "In", flux u 1:5 w
    1 t "Out"
if(doplot) set title "Total_flux,_ions"
if(doplot) plot flux u 1:6 w l t "In", flux u 1:7 w
    1 t "Out"
if(doplot) set key off
if(doplot) set title "Net_charge_change"
if(doplot) plot flux u 1:8 w l
if(doplot) unset multiplot

f='../diag/flux2.dat'
set xlabel "t_{/Symbol_w}_{pe}^{-1}]"
unset ylabel

```

```

set multiplot layout 2,1 title tittel
set key center left box width -3
set title 'Electrons'
plot f u 1:2 w l t 'In_acc', //
    f u 1:3 w l t 'Out_acc_x=0', //
    f u 1:4 w l t 'Out_acc_x=L', //
    f u 1:5 w l t 'In_refl', //
    f u 1:6 w l t 'Out_refl_x=0', //
    f u 1:7 w l t 'Out_refl_x=L' //
set key center left box width -3
set title 'Ions'
plot f u 1:8 w l t 'In_acc', //
    f u 1:9 w l t 'Out_acc_x=0', //
    f u 1:10 w l t 'Out_acc_x=L', //
    f u 1:11 w l t 'In_refl', //
    f u 1:12 w l t 'Out_refl_x=0', //
    f u 1:13 w l t 'Out_refl_x=L' //
unset multiplot
set autoscale

```

### A.2.9 results.vdf.p

```

time0="'cat_marks/time0'"
time1="'cat_marks/time1'"
time2="'cat_marks/time2'"
time3="'cat_marks/time3'"
time4="'cat_marks/time4'"
time5="'cat_marks/time5'"
time6="'cat_marks/time6'"
time7="'cat_marks/time7'"
time8="'cat_marks/time8'"
time9="'cat_marks/time9'"
time10="'cat_marks/time10'"

vhe0="'cat_marks/vdfe0'"
vhe1="'cat_marks/vdfe1'"
vhe2="'cat_marks/vdfe2'"
vhe3="'cat_marks/vdfe3'"
vhe4="'cat_marks/vdfe4'"
vhe5="'cat_marks/vdfe5'"
vhe6="'cat_marks/vdfe6'"
vhe7="'cat_marks/vdfe7'"
vhe8="'cat_marks/vdfe8'"
vhe9="'cat_marks/vdfe9'"
vhe10="'cat_marks/vdfe10'"

vhp0="'cat_marks/vdfp0'"
vhp1="'cat_marks/vdfp1'"
vhp2="'cat_marks/vdfp2'"
vhp3="'cat_marks/vdfp3'"
vhp4="'cat_marks/vdfp4'"
vhp5="'cat_marks/vdfp5'"
vhp6="'cat_marks/vdfp6'"
vhp7="'cat_marks/vdfp7'"
vhp8="'cat_marks/vdfp8'"
vhp9="'cat_marks/vdfp9'"
vhp10="'cat_marks/vdfp10'"

set xlabel 'v_{th}'
set multiplot layout 2,2 rowsfirst title tittel

set key off
options = "using_1:2_with_histeps_title"
set title "Electrons_x=lx"
plot vhe0 @options @time0, vhe1 @options @time1, vhe2
@options @time2, vhe3 @options @time3, vhe4
@options @time4, vhe5 @options @time5, vhe6
@options @time6, vhe7 @options @time7, vhe8
@options @time8, vhe9 @options @time9, vhe10
@options @time10

set key outside left center box samplen 2 width 0 title 't_{/'
Symbol_w}_{pe}^{-1}}'
options = "using_1:5_with_histeps_title"
set title "Electrons_x=ux"
plot vhe0 @options @time0, vhe1 @options @time1,
vhe2 @options @time2, vhe3 @options @time3,
vhe4 @options @time4, vhe5 @options @time5,
vhe6 @options @time6, vhe7 @options @time7,
vhe8 @options @time8, vhe9 @options @time9,
vhe10 @options @time10

set key off
options = "using_1:2_with_histeps_title"
set title "Ions_x=lx"
plot vhp0 @options @time0, vhp1 @options @time1,
vhp2 @options @time2, vhp3 @options @time3,
vhp4 @options @time4, vhp5 @options @time5,
vhp6 @options @time6, vhp7 @options @time7,
vhp8 @options @time8, vhp9 @options @time9,
vhp10 @options @time10

set key off
options = "using_1:5_with_histeps_title"
set title "Ions_x=ux"
plot vhp0 @options @time0, vhp1 @options @time1,
vhp2 @options @time2, vhp3 @options @time3,
vhp4 @options @time4, vhp5 @options @time5,
vhp6 @options @time6, vhp7 @options @time7,
vhp8 @options @time8, vhp9 @options @time9,
vhp10 @options @time10

unset multiplot

```

```

psetitle1="Electrons_t='cat_marks/time6'_{/'Symbol_w}_{pe}
}^{-1}}"
psedata1="'cat_marks/psxe-ve6'"
psetitle2="Electrons_t='cat_marks/time10'_{/'Symbol_w}_{pe}
}^{-1}}"
psedata2="'cat_marks/psxe-ve10'"
psititle1="Ions_t='cat_marks/time6'_{/'Symbol_w}_{pe}^{-1}}"
psidata1="'cat_marks/psxp-vp6'"
psititle2="Ions_t='cat_marks/time10'_{/'Symbol_w}_{pe}^{-1}}"
psidata2="'cat_marks/psxp-vp10'"

set ylabel 'v_{th}'
set xlabel 'x_{/'Symbol_w}_{D}'
set multiplot layout 2,2 title tittel

set key off

set title psetitle1
plot psedata1 with image

set title psetitle2
plot psedata2 with image

set title psititle1
plot psidata1 with image

set title psititle2
plot psidata2 with image

unset multiplot

```

### A.2.11 results.power.p

```

unset multiplot

set ylabel "Power"
set xlabel "t_{/'Symbol_w}_{pe}^{-1}}"
set logscale y
set key inside left top box title "k"

set size 1,0.5
set multiplot layout 1,2

set title "Left"
set size 0.5,0.5
load '../diag/power.left.p'

set size 0.5,0.5
set title "Right"
load '../diag/power.right.p'

unset multiplot

unset size
unset logscale y

```

### A.2.10 results.phasespace.p